

Springer

Berlin

Heidelberg

New York

Barcelona

Hong Kong

London

Milan

Paris

Tokyo

Helmut Alt (Ed.)

Computational Discrete Mathematics

Advanced Lectures



Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editor

Helmut Alt
Freie Universität Berlin, Institut für Informatik
Takustr. 9, 14195 Berlin, Germany
E-mail:alt@inf.fu-berlin.de

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Computational discrete mathematics : advanced lectures / Helmut Alt (ed.). -
Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ;
Paris ; Tokyo : Springer, 2001
(Lecture notes in computer science ; 2122)
ISBN 3-540-42775-9

CR Subject Classification (1998):F.2, G.2, I.3.5

ISSN 0302-9743

ISBN 3-540-42775-9 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

<http://www.springer.de>

© Springer-Verlag Berlin Heidelberg 2001
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN 10839914 06/3142 5 4 3 2 1 0

Preface

In order to speed up doctoral education in Germany the “Deutsche Forschungsgemeinschaft” (DFG, German Research Association) in the late 1980s developed a new funding concept for graduate programs called “Graduiertenkollegs”. Groups of university teachers could join together and submit proposals for doctoral studies within certain areas of research. If funded, the program would supply scholarships for doctoral students including means for travel, literature, scientific guests, meetings, and schools. The scholarships would allow doctorands to concentrate on their doctoral studies without the usual teaching duties and to obtain their degree in a much shorter period of time.

It was the idea of *Martin Aigner* to join efforts of mathematicians and computer scientists in the newly reunified Berlin and to apply for a graduate program within discrete mathematics and theoretical computer science. The final proposal was submitted by researchers from the three universities of Berlin, Free University, Humboldt-University, and Technical University, and the then still existing Academy of Sciences of the German Democratic Republic. The specific areas covered were geometric pattern recognition, constructive approximation, complexity theory, combinatorial optimization, graph theory, computational combinatorics, coding theory, and group theory. The proposal was accepted by the DFG, the program was named “Algorithmische Diskrete Mathematik” (Computational Discrete Mathematics), and Emo Welzl was elected as its first speaker. The program started with the first four doctorands on October 1st, 1991.

The program was extended twice by the DFG and ended in September 2000 after the maximal possible runtime of nine years. Twenty-five of the students funded obtained their doctoral degree, most of them within a time period significantly below average and many of them even within the standard two and a half years of funding by the program. All in all during the last years, discrete mathematics and algorithmics in general have been flourishing within Berlin mainly due to the existence of the graduate program.

In order to enhance contact and cooperation between the various research areas of the members of the program it had been decided at the beginning that there should be a weekly meeting on Monday afternoons during semesters. Part of this meeting consists of a 60 to 90 minute tutorial lecture by faculty members of the program or selected guests. These lectures should be understandable not only to specialists but to all members of research groups involved in the program. Still, they should be at a high scientific level including recent research results within the different fields of the program. They have turned out to be very popular and very fruitful for all groups involved including the faculty members of the program who use the opportunity to learn about the research areas of their colleagues. In order to make the material of these lectures available to a larger public we created this booklet containing twelve selected lectures held within the graduate program. It covers the whole range of areas represented

in the program including combinatorics, graph theory, coding theory, discrete and computational geometry, optimization, and algorithmic aspects of algebra. Particularly intriguing are the nonobvious connections between different areas such as combinatorics, linear algebra, discrete geometry, and graph theory in the contribution by Martin Aigner; algebraic computing and graph theory in the one by Günter Rote; or discrete geometry, graph theory, and coding theory in the one by Günter M. Ziegler.

A preliminary version of this booklet was published internally on the day of celebration of the end of the old graduate program and the start of a new one called “Combinatorics, Geometry, and Computation” which is a European graduate program of the same research groups together with partners at ETH Zurich and other European institutions.

Berlin, June 1, 2001

Helmut Alt

Table of Contents

Lattice Paths and Determinants	1
<i>Martin Aigner (Freie Universität Berlin)</i>	
The Nearest Neighbor	13
<i>Helmut Alt (Freie Universität Berlin)</i>	
Explicit and Implicit Enforcing – Randomized Optimization	25
<i>Bernd Gärtner and Emo Welzl (ETH Zürich)</i>	
Codes over Z_4	47
<i>Tor Helleseht (University of Bergen)</i>	
Degree Bounds for Long Paths and Cycles in k -connected Graphs	56
<i>Heinz Adolf Jung (Technische Universität Berlin)</i>	
Data Structures for Boolean Functions	61
<i>Christoph Meinel and Christian Stangier (Universität Trier)</i>	
Scheduling under Uncertainty: Bounding the Makespan Distribution	79
<i>Rolf H. Möhring (Technische Universität Berlin)</i>	
Random Graphs, Random Triangle-Free Graphs, and Random Partial Orders	98
<i>Hans Jürgen Prömel and Anusch Taraz (Humboldt-Universität zu Berlin)</i>	
Division-Free Algorithms for the Determinant and the Pfaffian: Algebraic and Combinatorial Approaches	119
<i>Günter Rote (Freie Universität Berlin)</i>	
Check Character Systems and Anti-symmetric Mappings	136
<i>Ralph-Hardo Schulz (Freie Universität Berlin)</i>	
Algorithms in Pure Mathematics	148
<i>Gernot Stroth (Martin-Luther-Universität Halle Wittenberg)</i>	
Coloring Hamming Graphs, Optimal Binary Codes, and the 0/1-Borsuk Problem in Low Dimensions	159
<i>Günter M. Ziegler (Technische Universität Berlin)</i>	
Author Index	173

Author Index

Aigner, Martin	1	Rote, Günter	119
Alt, Helmut	13		
Gärtner, Bernd	25	Schulz, Ralph-Hardo	136
Helleseeth, Tor	47	Stangier, Christian	61
		Stroth, Gernot	148
Jung, Heinz Adolf	56		
Meinel, Christoph	61	Taraz, Anusch	98
Möhring, Rolf H.	79		
		Welzl, Emo	25
Prömel, Hans Jürgen	98	Ziegler, Günter M.	159

Lattice Paths and Determinants

Martin Aigner

Department of Mathematics and Computer Science, Free University of Berlin
Arnimallee 3, 14195 Berlin, Germany
`aigner@math.fu-berlin.de`

The essence of mathematics is proving theorems – and so, that is what mathematicians do: they prove theorems. But to tell the truth, what they really want to prove once in their lifetime, is a *Lemma*, like the one by Fatou in analysis, the Lemma of Gauss in number theory, or the Burnside-Frobenius Lemma in combinatorics.

Now what makes a mathematical statement a true Lemma? First, it should be applicable to a wide variety of instances, even seemingly unrelated problems. Secondly, the statement should, once you have seen it, be completely obvious. The reaction of the reader might well be one of faint envy: Why haven't I noticed this before? And thirdly, on an esthetic level, the Lemma including its proof should be beautiful!

In this article we look at one such marvellous piece of mathematical reasoning, the Lemma of Gessel and Viennot which has become an instant classic in combinatorial enumeration since its appearance in 1985 [1]. A similar version which was largely overlooked was previously obtained by Lindström [2].

1 Four Problems

Let us consider the following four problems which do not appear to have much in common – and yet they can all be solved by the Gessel–Viennot Lemma.

A. The following curious problem was posed by Berlekamp and answered by Carlitz, Roselle and Scoville, see [4], p. 231. Suppose we are given a Young diagram D corresponding to the partition $\lambda : n_1 n_2 \dots n_t, n_1 \geq n_2 \geq \dots \geq n_t$. Given a square s of D , let ℓ be the lowest square in the same column as s , and r the rightmost square in the same row as s . Insert into s the number $f(s)$ of paths from ℓ to r which stay inside D and use steps to the north and east only.

Figure 1 shows the diagram of $\lambda : 765532$ and the numbers $f(s)$. Consider the largest square matrix M contained in D . In the figure, the 4×4 -matrix is drawn with bold lines.

Problem. *Prove that always $\det M = 1$.*

B. Everybody knows the Catalan numbers C_n , defined by the convolution recursion

$$C_0 = 1, \quad C_{n+1} = \sum_{k=0}^n C_k C_{n-k} \quad (n \geq 0).$$

142	46	16	7	2	1	1
47	16	6	3	1	1	
19	7	3	2	1		
5	2	1	1	1		
2	1	1				
1	1					

Fig. 1.

Stanley lists in his admirable book [4] some 70 combinatorial instances which are counted by the Catalan numbers. One of the classical examples are *Dyck paths* of length $2n$. These are lattice paths from $(0,0)$ to $(2n,0)$ which never go below the line $y = 0$ and use only diagonal steps north-east or south-east. The number of these Dyck paths (of length $2n$) is precisely C_n .

The next figure shows the $C_3 = 5$ Dyck paths of length 6:

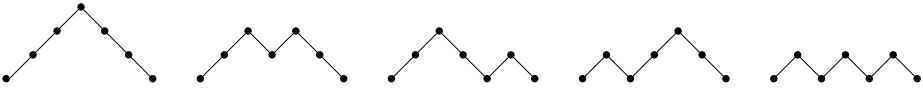


Fig. 2.

One of the most elegant characterizations of the sequence (C_n) goes as follows: Associate to (C_n) the Hankel matrices H_n and \tilde{H}_n ($n \geq 0$), where

$$H_n = \begin{pmatrix} C_0 & C_1 & \dots & C_n \\ C_1 & C_2 & \dots & C_{n+1} \\ & & \dots & \\ C_n & C_{n+1} & \dots & C_{2n} \end{pmatrix}, \quad \tilde{H}_n = \begin{pmatrix} C_1 & C_2 & \dots & C_{n+1} \\ C_2 & C_3 & \dots & C_{n+2} \\ & & \dots & \\ C_{n+1} & C_{n+2} & \dots & C_{2n+1} \end{pmatrix}.$$

Of course, we may associate such a pair of Hankel matrices to *any* sequence of real numbers.

Problem. Prove that (C_n) is the unique sequence of real numbers such that

$$\det H_n = \det \tilde{H}_n = 1 \quad \text{for all } n \geq 0.$$

C. The *Stirling numbers of the second kind* $S_{n,k}$ are defined as the number of partitions of an n -set into k blocks. A classical result states that the numbers $S_{n,k}$ satisfy the recursion

$$S_{0,0} = 1, \quad S_{0,k} = 0 \quad (k > 0)$$

$$S_{n,k} = S_{n-1,k-1} + kS_{n-1,k} \quad (n \geq 1).$$

Problem. Prove that for any $m \geq 0, n \geq 1$:

$$\det \begin{pmatrix} S_{m+1,1} & S_{m+1,2} & \cdots & S_{m+1,n} \\ S_{m+2,1} & S_{m+2,2} & \cdots & S_{m+2,n} \\ & & \cdots & \\ S_{m+n,1} & S_{m+n,2} & \cdots & S_{m+n,n} \end{pmatrix} = (n!)^m.$$

D. A particularly appealing counting result (due to Mac Mahon [3]) concerns the enumeration of rhombic tilings of a regular hexagon of side length n . Consider the hexagon in figure 3 (of side length 3) and triangulate it. A rhombus consists of two triangles with a common side, and what we are interested in is the total number $h(n)$ of partitions (tilings) of the hexagon into rhombi.

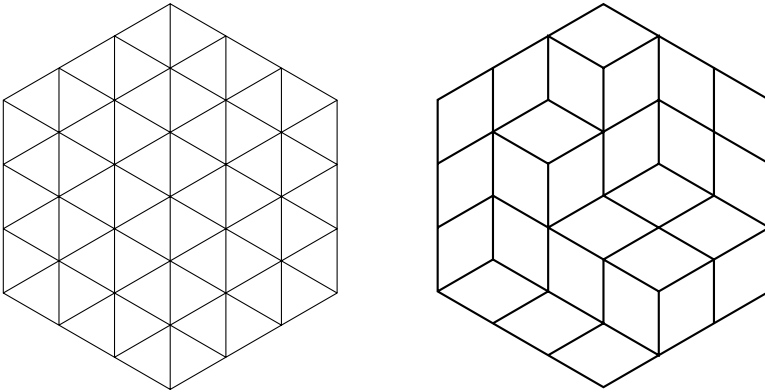


Fig. 3. A hexagon of length 3 and a rhombic tiling.

The figure on the right shows one possible tiling. One easily computes $h(1) = 2$, $h(2) = 20$. But already the case $n = 3$ becomes quite difficult to do by hand (in fact, $h(3) = 980$).

Problem. Find a formula for $h(n)$.

2 The Lemma of Gessel and Viennot

The starting point is the usual permutation representation of the determinant of a matrix. Let $M = (m_{ij})$ be a real $n \times n$ -matrix. Then

$$\det M = \sum_{\sigma} (\text{sign } \sigma) m_{1\sigma(1)} m_{2\sigma(2)} \cdots m_{n\sigma(n)}, \quad (1)$$

where σ runs through all permutations of $\{1, 2, \dots, n\}$, and the sign of σ is 1 or -1 depending on whether σ is the product of an even or odd number of transpositions.

Now we pass to graphs, more precisely to *weighted directed bipartite* graphs. Let the vertices A_1, \dots, A_n stand for the rows of M , and B_1, \dots, B_n for the columns. For all i and j draw an arrow from A_i to B_j and give it the weight m_{ij} as in the figure:

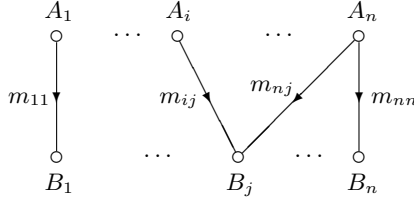


Fig. 4.

In terms of the graph, formula (1) has the following interpretation:

- The left-hand side is the determinant of the *path-matrix* M where the (i, j) -entry of M is the *weight* of the (unique) path from A_i to B_j .
- The right-hand side is the *weighted* (signed) *sum* over all *vertex-disjoint path systems* from $\mathcal{A} = \{A_1, \dots, A_n\}$ to $\mathcal{B} = \{B_1, \dots, B_n\}$. Such a system \mathcal{P}_σ is given by paths

$$\mathcal{P}_\sigma : A_1 \rightarrow B_{\sigma(1)}, \dots, A_n \rightarrow B_{\sigma(n)} ,$$

and the *weight* of \mathcal{P}_σ is the product of the weights of the individual paths:

$$w(\mathcal{P}_\sigma) = w(A_1 \rightarrow B_{\sigma(1)}) \cdots w(A_n \rightarrow B_{\sigma(n)}) .$$

In this interpretation formula (1) reads

$$\det M = \sum_{\sigma} (\text{sign } \sigma) w(\mathcal{P}_\sigma) .$$

And what is the result of Gessel and Viennot? It is the natural generalization of (1) from bipartite to *arbitrary weighted directed graphs*. It is precisely this step to arbitrary graphs that makes the Lemma so widely applicable – and what's more, the proof is stupendously simple and elegant.

Let us first collect the necessary concepts. We are given a finite acyclic directed graph $G = (V, E)$. In particular, there are only finitely many directed paths between any two vertices A and B , where we include all trivial paths $A \rightarrow A$ of length 0. Every edge e carries a real weight $w(e)$. If P is a directed path from A to B , written shortly $P : A \rightarrow B$, then we define the *weight* of P as

$$w(P) = \prod_{e \in P} w(e)$$

$$w(P) = 1 \quad \text{if } P \text{ has length } 0 .$$

Now let $\mathcal{A} = \{A_1, \dots, A_n\}$ and $\mathcal{B} = \{B_1, \dots, B_n\}$ be two sets of n vertices, where \mathcal{A} and \mathcal{B} need not be disjoint. To \mathcal{A} and \mathcal{B} we associate the *path matrix* $M = (m_{ij})$ with

$$m_{ij} = \sum_{P: A_i \rightarrow B_j} w(P) .$$

A *path system* \mathcal{P} from \mathcal{A} to \mathcal{B} consists of a permutation σ together with n paths $P_i : A_i \rightarrow B_{\sigma(i)} (i = 1, \dots, n)$; we write $\text{sign } \mathcal{P} = \text{sign } \sigma$. The *weight* of \mathcal{P} is the product of the path weights

$$w(\mathcal{P}) = \prod_{i=1}^n w(P_i) . \quad (2)$$

Finally, we say that the path system $\mathcal{P} = (P_1, \dots, P_n)$ is *vertex-disjoint* if the paths of \mathcal{P} are pairwise vertex-disjoint.

LEMMA. *Let $G = (V, E)$ be a finite acyclic weighted directed graph, $\mathcal{A} = \{A_1, \dots, A_n\}$ and $\mathcal{B} = \{B_1, \dots, B_n\}$ two n -sets of vertices, and M the path matrix from \mathcal{A} to \mathcal{B} . Then*

$$\det M = \sum_{\mathcal{P} \text{ vertex-disjoint}} (\text{sign } \mathcal{P}) w(\mathcal{P}) . \quad (3)$$

Proof. A typical summand of $\det M$ is

$$\begin{aligned} (\text{sign } \sigma) m_{1\sigma(1)} \cdots m_{n\sigma(n)} = \\ (\text{sign } \sigma) \left(\sum_{P_1: A_1 \rightarrow B_{\sigma(1)}} w(P_1) \right) \cdots \left(\sum_{P_n: A_n \rightarrow B_{\sigma(n)}} w(P_n) \right) . \end{aligned}$$

Summing over σ we immediately find from (2) that

$$\det M = \sum_{\mathcal{P}} (\text{sign } \mathcal{P}) w(\mathcal{P}) ,$$

where \mathcal{P} runs through *all* path systems from \mathcal{A} to \mathcal{B} (vertex-disjoint or not). Hence to arrive at (3), all we have to show is

$$\sum_{\mathcal{P} \in N} (\text{sign } \mathcal{P}) w(\mathcal{P}) = 0 , \quad (4)$$

where N is the set of all path systems that are *not* vertex-disjoint. And this is accomplished by an argument of singular beauty. Namely, we exhibit an involution $\pi : N \rightarrow N$ (without fixed points) such that for \mathcal{P} and $\pi\mathcal{P}$

$$w(\pi\mathcal{P}) = w(\mathcal{P}) \text{ and } \text{sign } (\pi\mathcal{P}) = -\text{sign } \mathcal{P} .$$

Clearly, this will imply (4) and thus formula (3).

The involution π is defined in the most natural way. Let $\mathcal{P} \in N$ with paths $P_i : A_i \rightarrow B_{\sigma(i)}$. By definition, some pair of paths will intersect:

- Let i_0 be the minimal index such that P_{i_0} shares some vertex with another path.
- Let X be the first such common vertex on the path P_{i_0} .
- Let $j_0 (> i_0)$ be the minimal index such that P_{j_0} has the vertex X in common with P_{i_0} .

The figure shows the situation:

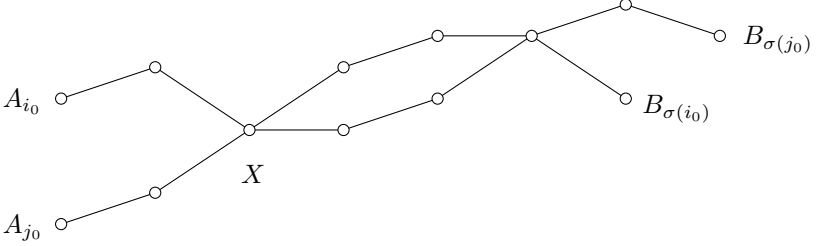


Fig. 5.

Now we construct the new system $\pi\mathcal{P} = (P'_1, \dots, P'_n)$ as follows:

- $P'_k = P_k$ for all $k \neq i_0, j_0$,
- the new path P'_{i_0} goes from A_{i_0} to X along P_{i_0} , and then continues to $B_{\sigma(j_0)}$ along P_{j_0} . Similarly, P'_{j_0} goes from A_{j_0} to X along P_{j_0} and continues to $B_{\sigma(i_0)}$ along P_{i_0} .

Clearly $\pi(\pi\mathcal{P}) = \mathcal{P}$, since the index i_0 , the vertex X , and the index j_0 are the same as before. In other words, applying π twice we switch back to the old paths P_i . Next, since $\pi\mathcal{P}$ and \mathcal{P} use precisely the same edges, we certainly have $w(\pi\mathcal{P}) = w(\mathcal{P})$. And finally, since the new permutation σ' is obtained by multiplying σ with the transposition (i_0, j_0) , we find $\text{sign}(\pi\mathcal{P}) = -\text{sign} \mathcal{P}$, and that's it.

3 A Graphical Approach to Determinants

Before going on to the problems, let us see how the Lemma of Gessel-Viennot can be used to derive all basic properties of determinants, just by looking at appropriate graphs. We consider four examples.

1. $\det M^T = \det M$.

Just look at figure 4 from bottom up. In other words, we reverse all arrows. Since $\text{sign} \sigma^{-1} = \text{sign} \sigma$, the result follows.

2. If the rows A_1, \dots, A_n of M are linearly dependent, then $\det M = 0$.

Suppose $A_n = \lambda_1 A_1 + \dots + \lambda_{n-1} A_{n-1}$, then the following graph has M as path matrix from \mathcal{A} to \mathcal{B} : Put A_n on top with an arrow of weight λ_i leading to A_i for all i , and keep the rest.

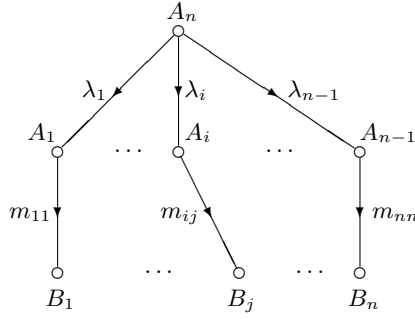


Fig. 6.

Since in this graph there is no vertex-disjoint path system from \mathcal{A} to \mathcal{B} at all, we infer $\det M = 0$ from (3).

3. $\det(MM') = (\det M) (\det M')$.

Let the bipartite graph on \mathcal{A} and \mathcal{B} correspond to M as in figure 4, and similarly the bipartite graph on \mathcal{B} and \mathcal{C} to M' . Consider now the concatenated graph as in figure 7. Since $(MM')_{ij} = \sum_k m_{ik} m'_{kj}$, we find that the path matrix from \mathcal{A} to \mathcal{C} is precisely the product MM' .

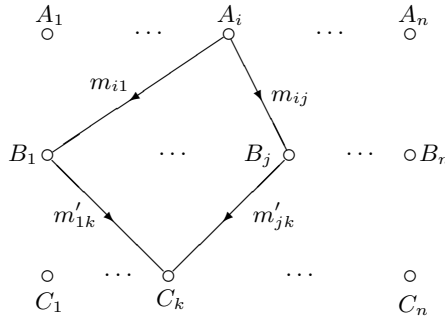


Fig. 7.

As the vertex-disjoint path systems from \mathcal{A} to \mathcal{C} in the concatenated graph correspond to pairs of systems from \mathcal{A} to \mathcal{B} resp. from \mathcal{B} to \mathcal{C} , the result follows immediately from the Lemma, noting that $\text{sign}(\sigma\tau) = (\text{sign } \sigma)(\text{sign } \tau)$.

4. Formula of Binet-Cauchy

A very useful generalization of the product rule is a result named after Binet and Cauchy. Let M be a $n \times p$ -matrix and M' a $p \times n$ -matrix, $n \leq p$. Then

$$\det(MM') = \sum_Z (\det M_Z) (\det M'_Z),$$

where M_Z is the $n \times n$ -submatrix of M with columns Z , and M'_Z the submatrix of M' with the corresponding rows Z .

The proof proceeds as in 3) by looking at the corresponding concatenated graph.

4 Solving the Problems

One of the nicest features of the Lemma is the fact that formula (3) can be read in two ways. It gives a method to compute a *determinant* if we can evaluate the sum on the right-hand side of (3). But it can also be used to solve an *enumeration problem* (interpreted as the right-hand side of (3)) if we can somehow calculate the determinant on the left. Problems A), B) and C) will illustrate the first aspect, and problem D) the second.

A. Let D be the Young diagram of $\lambda : n_1 n_2 \dots n_t$. We construct a directed graph G_D as follows:

The vertices are the squares of D , and we draw an edge (of weight 1) between any two neighbouring squares in the direction north or east. Suppose the largest square array contained in D has size $n \times n$. Then we take A_1, \dots, A_n to be the bottom squares of columns $1, \dots, n$, and B_1, \dots, B_n the rightmost squares of rows $1, \dots, n$.

As an illustration, the graph G_D corresponding to the diagram in figure 1 looks as follows:

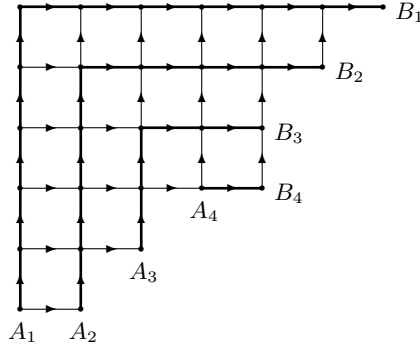


Fig. 8.

Since all weights are equal to 1, the matrix M (filled with the numbers $f(s)$) is clearly the path matrix from \mathcal{A} to \mathcal{B} , so to apply the Lemma, we have to find all vertex-disjoint path systems from \mathcal{A} to \mathcal{B} . But a moment's thought should convince the reader that there is only *one* such system, connecting A_i with B_i ($i = 1, \dots, n$) in “hooks”. (They are drawn with bold lines in the figure.) As $\text{sign}(\text{id}) = 1$ and $w(\mathcal{P}) = 1$, we obtain $\det M = 1$ as claimed.

B. Note first that the sequence of determinants $\det H_0$, $\det \tilde{H}_0$, $\det H_1$, $\det \tilde{H}_1, \dots$ uniquely determines the original sequence provided that $\det H_n \neq 0$, $\det \tilde{H}_n \neq 0$ for all n . Hence if we can show $\det H_n = \det \tilde{H}_n = 1$ for the Catalan numbers C_n , then (C_n) will indeed be the only sequence with this property.

However, $\det H_n = 1$ resp. $\det \tilde{H}_n = 1$ for the Catalan numbers is just the special case of **A**) applied to the partitions $\lambda : 2n + 1, 2n, \dots, 1$ resp. $\tilde{\lambda} : 2n + 2, 2n + 1, \dots, 1$. The figure shows these graphs for $n = 2$ (with the arrows omitted):

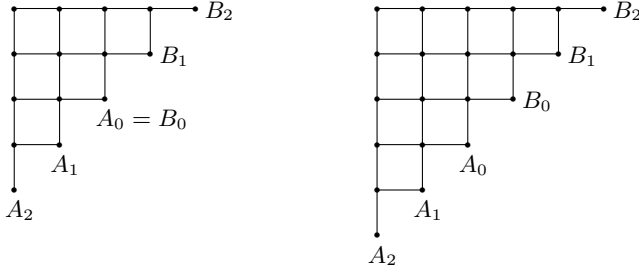


Fig. 9.

Tilting the Dyck paths in figure 2 by 45 degrees, it should be clear that, in general, the paths from A_i to B_j in figure 9 are just the Dyck paths of length $2(i + j)$ resp. $2(i + j + 1)$. Hence the path matrices are precisely $H_n = (C_{i+j})$ resp. $\tilde{H}_n = (C_{i+j+1})$, and the assertion follows from **A**).

C. Take the Stirling table $(S_{n,k})$ and turn it counterclockwise by 90° . That is, $S_{n,k}$ is placed at the point (n, k) . The recursion

$$S_{n,k} = S_{n-1,k-1} + kS_{n-1,k}$$

tells us the following. Consider the directed graph in the upper half plane $y \geq 0$, with edges going diagonally up (north-east) with weight 1 and horizontally (east) with weight k if the horizontal step occurs at $y = k$. Then $S_{n,k}$ is the sum of all weighted paths from $(0, 0)$ to (n, k) .

The figure shows $n = 5, k = 3$ with $S_{5,3} = 25$.

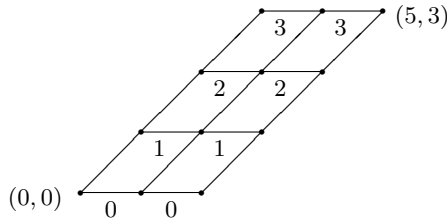


Fig. 10.

There are 6 non-zero paths from $(0, 0)$ to $(5, 3)$ with weights 1, 2, 3, 4, 6 and 9, summing to 25.

Hence to compute the determinant in problem **C)** we use the following set-up:

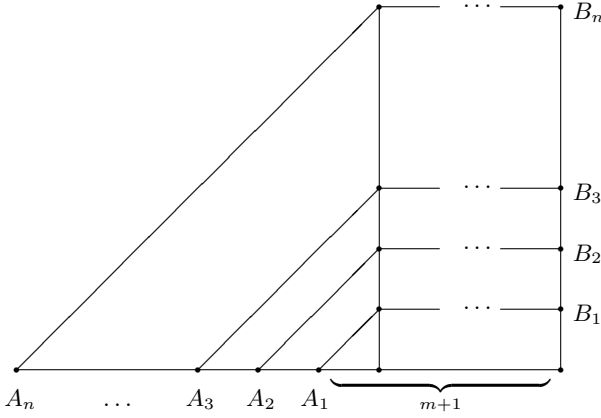


Fig. 11.

Defining the steps and weights as before we find that the sum of the weighted paths from A_i to B_j is precisely $S_{m+i,j}$ ($1 \leq i, j \leq n$). Hence the matrix of Stirling numbers is precisely the path matrix M of the graph in figure 11. And what are the vertex-disjoint path systems? Again there is only one, connecting A_i to B_i ($i = 1, \dots, n$) as in the figure. The diagonal weights are all 1, and of the horizontal steps there are m at every level $y = k$ ($k = 1, \dots, n$). So we conclude from the Lemma

$$\det M = 1^m 2^m \cdots n^m = (n!)^m .$$

D. Take a triangulated hexagon of side length n and associate to it the directed graph G as in figure 12 with all edges going up and all weights equal to 1.

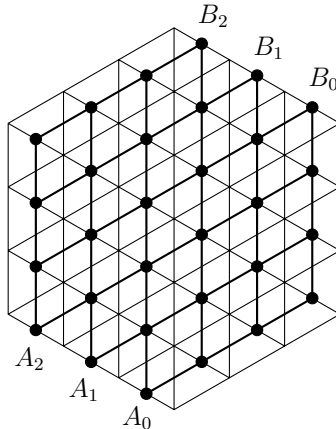


Fig. 12.

Let \mathcal{A} be the vertices A_0, A_1, \dots, A_{n-1} at the bottom and $\mathcal{B} = \{B_0, \dots, B_{n-1}\}$ those on top. It should be clear from the figure that any vertex-disjoint path system \mathcal{P} from \mathcal{A} to \mathcal{B} must join A_i with B_i for all i , that is $\text{sign } \mathcal{P} = 1$ for all \mathcal{P} .

Here is the key observation: The rhombic tilings are in one-to-one correspondence with the vertex-disjoint path systems from \mathcal{A} to \mathcal{B} . To see this, look at a rhombic tiling in a “3-dimensional” fashion, as suggested by figure 13.

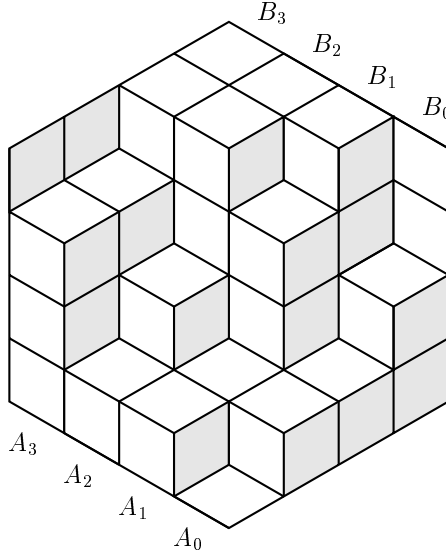



Fig. 13.

We shade all tiles of shape . Climbing up the “white ladders” yields a vertex-disjoint path system in G from \mathcal{A} to \mathcal{B} . Conversely, it is readily seen that any such system in G gives a tiling by adjoining “white” tiles according to the paths, and filling in the rest with shaded tiles.

So, what we have proved is that for this directed graph G

$$\sum_{\mathcal{P} \text{ vertex-disjoint}} w(\mathcal{P}) = h(n).$$

To apply the Lemma it remains to compute the number m_{ij} of paths in G from A_i to B_j , and then to evaluate $\det M$.

Now a moment’s reflection shows that m_{ij} really counts paths in a square grid (look at figure 12), and we find

$$m_{ij} = \binom{2n}{n+i-j} \quad (0 \leq i, j \leq n-1).$$

From this, the determinant of $M = (m_{ij})$ is easily evaluated, yielding the following beautiful formula of Mac Mahon:

$$h(n) = \prod_{i=0}^{n-1} \frac{\binom{2n+i}{n}}{\binom{n+i}{n}}.$$

As examples, we obtain again $h(1) = 2$, $h(2) = 20$, $h(3) = 980$, and as next value $h(4) = 232848$.

References

1. I. M. Gessel, G. Viennot: Binomial determinants, paths, and hook length formulae. *Advances in Math.* 58 (1985), 300–321.
2. B. Lindström: On the vector representation of induced matroids. *Bull. London Math. Soc.* 5 (1973), 85–90.
3. P. A. Mac Mahon: *Combinatory Analysis*, Vol. 2. Chelsea 1960.
4. R. P. Stanley: *Enumerative Combinatorics*, Vol. 2. Cambridge Univ. Press 1999.

The Nearest Neighbor

Helmut Alt

Department of Mathematics and Computer Science, Free University of Berlin,
Takustr.9, 14195 Berlin, Germany
alt@inf.fu-berlin.de

1 The Problem

The *nearest neighbor problem* is defined as follows:

Given a metric space X and a fixed finite subset $S \subset X$ of n “sites”, preprocess S and build a data structure so that queries of the following kind can be answered efficiently: Given a point $q \in X$ find one of the points $p \in S$ closest to q (see Figure 1).

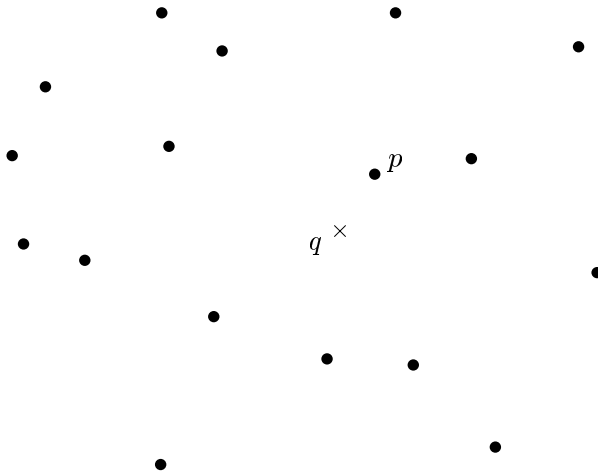


Fig. 1. The nearest-neighbor problem in the plane with Euclidean distance

Clearly, the question whether this problem can be solved efficiently, depends on the underlying metric space, here we will consider the d -dimensional real space \mathbb{R}^d usually together with the ℓ_1 , ℓ_∞ , or the ℓ_2 (Euclidean) metric. The criteria for the efficiency of a solution are the *preprocessing time* to construct the data structure, its *storage requirements*, and, in particular, the *query time*. Since preprocessing needs to be done only once, storage and query time are usually the more important criteria.

The nearest neighbor problem is a very natural problem in computational geometry and, therefore, interesting by itself. Originally, it was formulated by Minsky and Papert [24] and its two-dimensional version was posed in 1972 by McNutt in the context of finding the nearest post office from some given location q . Knuth [20] describes McNutt’s data structure, the post-office tree, and since then the problem is also known as the *post-office-problem*.

The nearest-neighbor problem also has numerous applications. In fact, nearest neighbor methods are being used in statistics and data analysis [6] information retrieval [27], data compression [12], pattern recognition [8] and multimedia databases [26].

2 The Voronoi-Diagram Approach

The structure in computational geometry that is most closely related to nearest-neighbor or other “proximity” problems is the *Voronoi-diagram* (for a survey see [3], [25]).

For a given set $S \subset \mathbb{R}^d$ the Voronoi-diagram $VD(S)$ is a partitioning of \mathbb{R}^d assigning to each site $p \in S$ its *Voronoi region* consisting of those points $x \in \mathbb{R}^d$ that are closer to p than to any other point in S . The Euclidean distance Voronoi diagram in two dimensions (see Figure 2) is well studied and understood. It has the following properties:

1. The Voronoi regions are convex and bounded by line segments or rays, which are called *Voronoi-edges*.
2. Each Voronoi edge is a connected segment of the *bisector* between two points $p, q \in S$ i.e. the straight line consisting of all points equally close to p and q .
3. The endpoints of the Voronoi edges are called *Voronoi vertices*. Each of them has at least three closest points in S , and, therefore is the center of the circumcircle of these points.
4. The Voronoi diagram can be considered as an embedded planar graph with n faces (adding an artificial Voronoi vertex “at infinity” which is an endpoint of all unbounded edges). It can be derived from Euler’s formula for planar graphs that the number of edges is at most $3n - 6$ and the number of vertices at most $2n - 5$. So altogether, the Voronoi diagram has a complexity linear in the number of sites.
5. The Voronoi diagram can be constructed in $O(n \log n)$ time. Various algorithms for this purpose have been found, for example a divide-and-conquer approach, a randomized incremental construction, a sweepline technique, and a reduction to computing convex hulls of finite sets of points in three dimensions (see e.g. [25], [18], [4]).

Clearly, once the Voronoi diagram of S has been computed, the nearest-neighbor problem reduces to a *point location* of the query point q within $VD(S)$, i.e. finding the Voronoi cell that contains q .

A very simple point location data structure that was already described in one of the first papers on nearest neighbors by Dobkin and Lipton [7] can be constructed as follows:

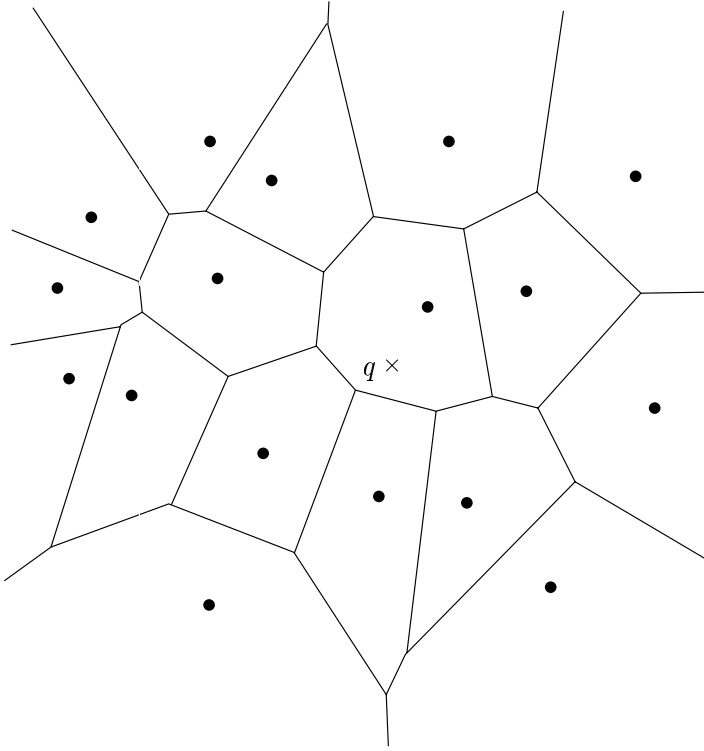


Fig. 2. A two-dimensional Euclidean distance Voronoi-diagram

Suppose (without loss of generality) that no Voronoi edges are vertical. Project all Voronoi vertices to the x -axis yielding $O(n)$ real intervals

$$(-\infty, x_1], [x_1, x_2] \dots [x_m, +\infty)$$

(see Figure 3). If we consider the vertical straight lines through all $x_i, i = 1, \dots, m$ we obtain a partition of the plane into vertical “slabs”. Each slab is subdivided into triangular or trapezoidal cells by the sequence of Voronoi-edges that cross it. For each slab we store this sequence ordered from bottom to top together with the information for each cell to which Voronoi region it belongs.

With this data structure a point location and thus a nearest neighbor query can be performed as follows:

- given the query point q , do a binary search of its x -coordinate within the intervals of the x -axis, to obtain the slab containing q .
- Within this slab, do a binary search on the ordered sequence of Voronoi edges to obtain the cell containing q . Then we also have the Voronoi region containing q and, thus, its nearest neighbor.

Since the processing of the query essentially consists of two binary searches on structures whose size is linear in n the query time is $O(\log n)$.

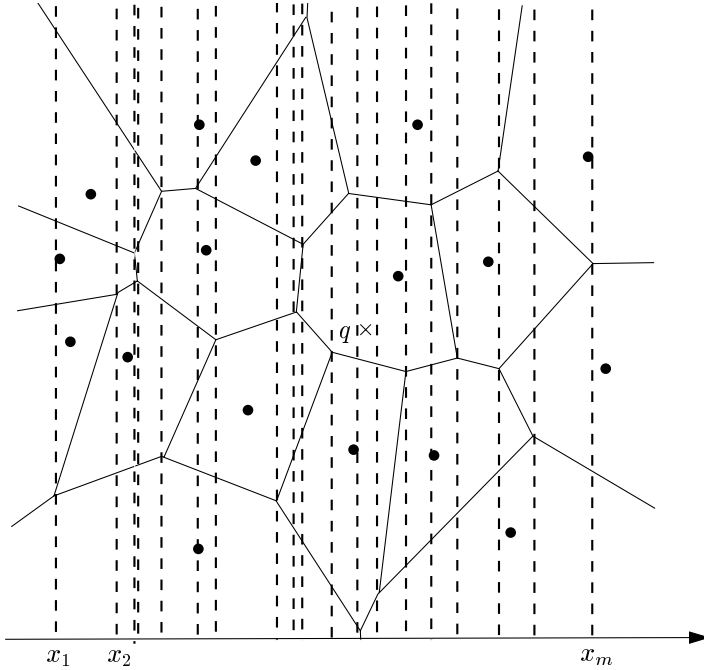


Fig. 3. The point location data structure

The preprocessing time is $O(n \log n)$ for constructing the Voronoi-diagram plus the time to construct the slab data structure. Since there are $O(n)$ slabs each containing $O(n)$ cells, the total storage required for this structure is $O(n^2)$ and it can be constructed in $O(n^2)$ time, as well.

Quadratic storage, however, is already quite high in applications. Fortunately, there are more efficient, although more complicated data structures, for planar point location. They use a hierarchical decomposition of the plane represented by balanced search-trees (see [17]). Applying them to our problem we can solve the nearest-neighbor-problem in two dimensions in $O(n \log n)$ preprocessing time, $O(n)$ storage, and $O(\log n)$ query time.

3 The Curse of Dimensionality

In many applications the nearest neighbor problem has to be solved in high-dimensional spaces. The techniques of the previous section can be generalized to higher dimensions, for example, the data structure of Dobkin and Lipton [7] and a Voronoi-diagram based structure by Clarkson [5]. For *fixed* dimensions these techniques have polynomial preprocessing time and storage requirements and $O(\log n)$ query time. However, the degrees of the polynomial and the constants in the O -terms depend on the dimension and, therefore, we should check how large the dimension can get in applications.

As a first example consider nearest-neighbor based *pattern recognition* (see e.g. [9]). Here the query pattern is usually given as a matrix of pixels with certain colors, grey-values, or black/white-values (as, for example, in character recognition). In a first step certain characteristic values called *features* of this pattern are extracted. These features can be ad-hoc like the ratio between the number of black and white pixels or the sum of grey-values in certain columns or they can be more sophisticated like the coefficients of certain frequencies in the discrete Fourier-transform of the pattern. These features are collected in a certain *feature vector* which should contain enough information to uniquely distinguish the pattern from others.

In fact, the feature vector is compared to a finite set of *prototypes* in a data structure. These prototypes are typical feature vectors for the different pattern classes like, for example, the different characters of the alphabet in character recognition. Now, among the prototypes the nearest neighbor of the feature vector of the query pattern is found which is propagated as the pattern class to which this pattern belongs.

In character recognition, for example, the dimension d in which the nearest-neighbor problem has to be solved is around 20 and the number n of different prototypes is about 100.

For a more concrete example let us consider the multimedia retrieval system *QBIC* (see [10]) which was developed at the IBM Almaden Research Center and operates on large image and video databases. A user can query the system for images or scenes, for example, in order to find all videos where a red car is moving from right to left with the camera zooming. The user can specify an image or scene by an explicit example image, a sketch or drawing, color or texture patterns, camera and object motion in the case of videos, or other graphical information. The system will then search the database for a sample of images or scenes that “come closest” to the user’s query. In particular, from the query a high-dimensional feature vector is constructed that contains information about color, texture, shape, motions, and relative positions of objects. With this vector a search is carried out among the feature vectors of the data base in order to find the closest ones. Essentially, we have a nearest-neighbor-problem where the dimension d of the space can be several hundreds and the number n of sites (=objects in the database) tens of thousands.

In other applications the dimension can be still higher and reach several thousands. If we consider, how the data structures mentioned before depend on the dimension, we find that they definitely cannot be used for these applications. In fact, Dobkin and Lipton’s first algorithm has preprocessing time $O(n^{2^{d+1}})$ and query time $O(2^d \log n)$ - and Clarkson’s algorithm has preprocessing time $O(n^{\lceil d/2 \rceil (1+\epsilon)})$ for any arbitrary fixed $\epsilon > 0$ and query time $O(2^d \log n)$. In fact, there is no hope that Voronoi-diagram based algorithms can do much better since the complexity (= number of facets of different dimensions) of the d -dimensional Voronoi-diagram is $O(n^{\lceil d/2 \rceil})$. The first result where at least the query time does not depend exponentially on the dimension was obtained by Meiser [23]. In

fact, the query time is $O(d^5 \log n)$ but the preprocessing time is still $O(n^{d+\epsilon})$ for any fixed $\epsilon > 0$.

All these techniques, although sophisticated are in higher dimensions not competitive with the *brute force solution*, where just the sites themselves are stored, i.e. we have “no” preprocessing time and the storage requirement is $O(nd)$. A query is answered by just determining the distances of the query point to all sites and extracting the minimum. This takes $O(nd)$ time, for example for all ℓ_p -distances, $1 \leq p \leq \infty$.

4 Approximate Solutions

In order to overcome the “curse of dimensionality” researchers started to look for approximate solutions for the nearest-neighbor-problem. For a constant $c \geq 1$ a *c-approximate nearest neighbor* of a query point q is an element x from the set S of sites which is not necessarily the nearest neighbor but its distance to q is at most by a factor c larger than the one of the nearest neighbor.

One of the early approximate solutions was given by Arya and Mount [2]. The obtained data structure for finding the $(1+\epsilon)$ -approximate nearest neighbor with preprocessing time $O(1/\epsilon)^d O(n)$ and query time $O(1/\epsilon)^d O(\log n)$. So there is still an exponential dependence on the dimension where the base depends on the quality of the approximation. The problem was investigated intensely in the last years and a real break-through was achieved by working with *randomized* techniques [19], [15], [21]. Preprocessing time, storage, and query time of the nearest neighbor data structures by Piotr Indyk [14] finally have no exponential dependence on the dimension any more.

In fact, for fixed $\epsilon > 0$ and the ℓ_1 -distance a $(1+\epsilon)$ -nearest-neighbor can be found with polynomial preprocessing time and storage and query time polynomial in $\log n$, d and $1/\epsilon$. This is a *Las Vegas-type* randomized algorithm, i.e. the result is always correct but the runtimes given are expected values and may be slower in the worst case. The results can be generalized to the ℓ_2 -distance, but then the approximation factor gets multiplied by $\sqrt{3}$.

For $\epsilon > 2$ the Las Vegas algorithm can be replaced by a deterministic one with similar efficiency bounds. The techniques for randomized approximate nearest-neighbor-search are quite sophisticated and we will here sketch the major ideas only.

The key idea is based on the surprising property of the nearest-neighbor-problem that for a fixed set S it is possible to project all points into a much lower-dimensional space without losing too much information about the proximity (i.e. closeness) of the points of S to other points. In fact, we have the following lemma by Johnson and Lindenstrauss (see [11]).

Lemma: *For any $\epsilon \in (0, 1/2)$ and any set $S \subset \mathbb{R}^d$, $|S| = n$, there exists a $k = O(1/\epsilon^2 \log n)$ and a linear map $f : S \rightarrow \mathbb{R}^k$ so that for all $u, v \in S$*

$$\|f(u) - f(v)\|^2 \in ((1 - \epsilon)\|u - v\|^2, (1 + \epsilon)\|u - v\|^2)$$

In fact, it can be shown that the projection into the subspace spanned by k randomly chosen unit vectors satisfies the lemma with high probability. In order to obtain the algorithms described above, the lemma of Johnson and Lindenstrauss is not used directly to reduce the dimension of the search space but it is combined with some other techniques.

In particular, nearest neighbor search is reduced to point location in equal balls (*PLEB*). This problem is defined as follows:

Given a set S on n sites, $(r, R) - PLEB$, $0 < r < R$, means to find out for a given query point q whether it lies in some ball of radius r around one of the sites. In this case the answer should be YES. If q does not lie in any ball of radius R around one of the sites the answer should be NO, otherwise it can be either YES or NO.

Somehow, *PLEB* is the decision problem corresponding to the nearest-neighbor computation problem (“Does the nearest neighbor have distance at most r ?”) and, therefore, once an efficient data structure for $(r, R) - PLEB$ with $R \leq (2 + O(\epsilon))r$ has been found, the approximate nearest neighbor search can be reduced to it by binary search.

Another technique used in Indyk’s method is the embedding of the finite metric space S with the d -dimensional ℓ_1 -metric into a *Hamming space*. The m -dimensional Hamming space consists of the finite set $\{0, 1\}^m$ with the Hamming metric, i.e. the distance between two $\{0, 1\}$ -vectors is the number of coordinates where they differ. It has been shown in [22] that an embedding of S into $\{0, 1\}^m$ is possible where the distance between points is distorted by at most a factor of $1 + \epsilon$ with $m = O(d \cdot \log n \cdot 1/\epsilon)$.

Altogether, Indyk’s method consists of the following 4 steps of reduction (NNS means “nearest-neighbor-search”, ℓ_i^d is the d -dimensional metric space with ground set S and distance ℓ_i , $i = 1, 2$):

$$\begin{aligned}
 & (\sqrt{3} + \epsilon) - \text{approximate NNS in } \ell_2^d \\
 & \quad \downarrow 1 \\
 & (1 + \epsilon) - \text{approximate NNS in } \ell_1^d \\
 & \quad \downarrow 2 \\
 & \quad PLEB \text{ in } \ell_1^d \\
 & \quad \downarrow 3 \\
 & \quad PLEB \text{ in } m - \text{dimensional Hamming-space} \\
 & \quad \downarrow 4 \\
 & PLEB \text{ in } k - \text{dimensional Hamming-space with } k = O(\log m).
 \end{aligned}$$

Step 1 is based on the fact that for any fixed $\epsilon > 0$, any \mathbb{R}^d with metric ℓ_p can be embedded into $\mathbb{R}^{O(d)}$ with metric ℓ_1 with distance distortion at most $1 + \epsilon$, which was shown by Johnson and Schechtman [16].

Steps 2 and 3 have been described before.

Step 4 is the key-step of the algorithm giving an explicit dimension-reduction in Hamming-space. It consists of several substeps.

First *hashing* is used to distribute the bits of a vector $x \in \{0, 1\}^m$ into p different buckets, where p is some suitably chosen prime. The bit sequences occurring within the buckets, when the vectors within the set S are hashed, are interpreted as symbols of some alphabet Σ . This process is repeated for several hash functions and the results are concatenated. The resulting function $f : \{0, 1\}^m \rightarrow \Sigma^r$ has the property that it does not distort distances too much. This effect is further enhanced by encoding the symbols of Σ by an *error correcting binary code*.

Alltogether, a function $g : \{0, 1\}^m \rightarrow \{0, 1\}^{d'}$ is obtained where d' depends logarithmically on n and m . In a further step the set of coordinates $\{1, \dots, d'\}$ is randomly partitioned into sets I_1, \dots, I_t of size $s = O(\log n)$. It can be shown that for all $x, y \in S$ which have a large distance the distances of all projections of $g(x)$ and $g(y)$ onto the subspaces given by the coordinate sets I_i are still large enough. This ensures that no serious error can occur when nearest neighbor search is done in the lower-dimensional subspaces.

The *PLEB* problem is finally solved by building a nearest-neighbor data structure for each set S_i obtained from S by projection onto the subspace given by I_i . Since I_i is $O(\log n)$ -dimensional this data structure is just a table telling for each point in $\{0, 1\}^s$ its nearest neighbor in S_i .

Although the analysis of Indyk's algorithm is quite complicated, the computations required in the single steps are not, so that the algorithm is possibly of practical significance.

5 Brute Force Revisited

Finally, we will present an algorithm for solving the nearest-neighbor problem with respect to the ℓ_∞ -distance. It has the advantage that it solves the problem exactly and, furthermore, is very simple. Therefore, it is easy to implement and has a small constant in the O -term of its asymptotic runtime. Like the brute-force method it requires no preprocessing and also only the point set S itself is stored. Its average runtime assuming that the set S is drawn randomly from the unit cube $[0, 1]^d$ under uniform distribution is $\Theta(nd/\ln n)$ thereby improving the brute-force method by a factor of $\Theta(1/\ln n)$. Experiments show that for

practical values of d and n our algorithm is up to 10 times faster than the brute-force method.

Throughout this section we will assume that $S = \{p_1, \dots, p_n\}$ is a fixed set of n points $p_i = (p_{i1}, \dots, p_{id})$ in \mathbb{R}^d . Since we can translate and scale any given set S without changing the problem we will assume without loss of generality that $S \subset [0, 1]^d$. For our probabilistic considerations we will assume that S is drawn at random from $[0, 1]^d$ under uniform distribution. A generalization to other “well-behaved” distributions is possible [13].

The idea of our algorithm is to consider a cube C of sidelength α around the query point q which is expected to contain a low number $\varphi(n)$ of the points in S . Suitable values of α and φ will be determined later. C will be chosen such that it contains q ’s nearest neighbor if it contains any point of S . As soon as a point $p \in S$ turns out not to lie in C the algorithm eliminates it from further consideration. This search algorithm will be called the *cube-method*. Following we will give a schematic description and fill in the details later.

The Cube-Method

input: a query point $q \in [0, 1]^d$

- (1) Determine a suitable number φ and an $\alpha > 0$, so that the expected number of points of P within the d -dimensional cube C with center q and sidelength α is φ .
- (2) **for** $i := 1$ **to** n **do** /* check point p_i */
- (3) $\{j := 1;$
- (4) **while** $(j \leq d \text{ and } |p_{ij} - q_j| \leq \alpha/2)$ **do** $j := j + 1;$
- (5) **if** $j = d + 1$ /* $d_{\max}(p_i, q) \leq \frac{\alpha}{2}$ */
- (6) **then** $\{\hat{p} := p_i ; \alpha := 2 \cdot d_{\max}(\hat{p}, q); \}$
- (7) **if** \hat{p} is defined **then** return \hat{p} as nearest neighbor
- (8) **else** determine the nearest neighbor by the brute-force method.

For the analysis of the cube-method let us assume for simplicity that the cube C is completely contained in $[0, 1]^d$. A detailed consideration [1] shows that the analysis holds in the general case, as well.

Since we assume that the set S is randomly drawn from $[0, 1]^d$ under uniform distribution, the expected number of points in cube C will be the product of its volume with n . So in order to get the value φ the volume of C should be φ/n and consequently the side length

$$\alpha = \left(\frac{\varphi}{n}\right)^{\frac{1}{d}} \quad (1)$$

should be chosen in step (1). Observe that the side length of C keeps changing in step (6) but it is always bounded by the original α in step (1). Because of the uniform distribution, the probability in step (4) that the j -th coordinate p_{ij} lies in the interval $[q_j - \frac{\alpha}{2}, q_j + \frac{\alpha}{2}]$ is α . Since these events are independent for different values of j , the probability that the while-loop in (4) is carried out at

least j times ($1 \leq j \leq d$) is α^j . Consequently, the expected number of tests performed in step (4) per point of P is

$$1 + \alpha + \alpha^2 \dots + \alpha^{d-1} \leq \frac{1}{1 - \alpha} \quad (2)$$

so the total expected number of tests in (4) is bounded by $n/(1 - \alpha)$.

Taking logarithms in (1) gives

$$\ln \alpha = \frac{\ln(\varphi/n)}{d} \quad (3)$$

Since for $\alpha \geq 1/3$, which we may assume for high dimensions, we have

$$1 - \alpha > -\frac{1}{2} \ln \alpha, \quad (4)$$

the expected number of executions of step (4) is at most

$$2 \frac{nd}{\ln(n/\varphi)}, \quad (5)$$

which gives already the desired time bound for steps (1) to (7) of the algorithm. However, there is a nonzero probability that the box C contains no points of P at all. In this case, the brute-force method will be called in step (8) having a runtime of $\Theta(nd)$. We will now determine the parameter φ such that the probability of this event is so small that it does not effect the total asymptotic runtime.

In fact, the probability that no point of P is in box C , i.e. that each point is in the complement of C is

$$\left(1 - \frac{\varphi}{n}\right)^n \quad (6)$$

As is well known, this expression is monotone increasing in n and converges to $e^{-\varphi}$ for $n \rightarrow \infty$. Hence, it is sufficient to choose φ such that the expected asymptotic runtime of step (8)

$$e^{-\varphi} \cdot nd \leq \frac{nd}{\ln(n/\varphi)} \quad (7)$$

so that it does not become worse than the runtime of the rest of the algorithm (see (5)). (7) is equivalent to $\varphi \geq \ln \ln(n/\varphi)$ which is satisfied, if we choose

$$\varphi \geq \ln \ln n \quad (8)$$

Another possibility would be not to invoke the brute force method in step (8) but to increase the volume of C and to start again. This method has been analyzed and it has the same asymptotic complexity (see [13]). However, the constant is better since for φ a constant can be chosen which decreases the initial value of α .

Some modifications of our algorithm are possible:

We did not analyze the effect of decreasing α in step (6). So the analysis also is correct, if we do not decrease α and at the end just compute by brute force the nearest neighbor among the points that are left in the cube of size α .

A detailed analysis shows that the decrease of α only influences the runtime by a constant factor. On the other hand, we even could omit step (1) and start with $\alpha = 1$ and get an $O\left(\frac{nd}{\ln n}\right)$ runtime [1].

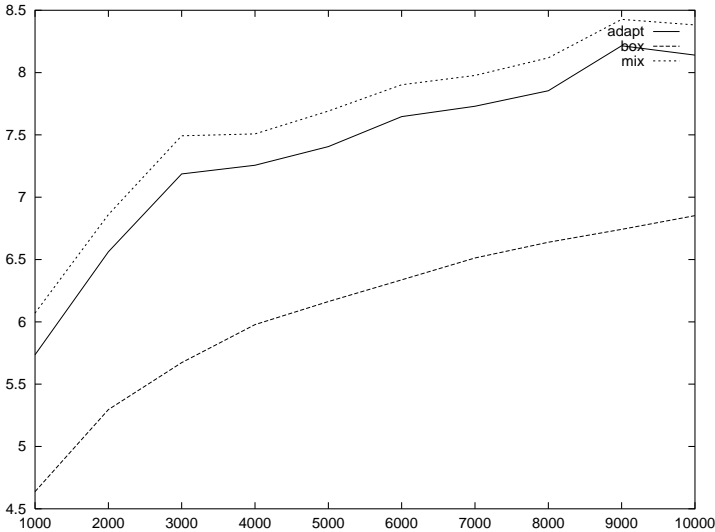


Fig. 4. Experimental comparison of three variants of our algorithm with the brute-force method

Figure 4 shows an experimental comparison of implementations of the three variants of the algorithm for dimension $d = 100$ and sizes of S between 1000 and 10000. The value on the vertical scale is the factor by which the algorithm is faster than the brute-force method. *box* is the variant without decreasing α , *adapt* is the variant with decreasing α starting with $\alpha = 1$, and *mix* is the mixture of both, i.e. the algorithm given before.

References

1. H. Alt, L. Heinrich-Litan, U. Hoffmann. Exact ℓ_∞ nearest neighbor search in high dimensions. Technical report, Institute for Computer Science, FU Berlin, in preparation.
2. S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 271–280, 1993.

3. F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, Sept. 1991.
4. J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, UK, 1998. Translated by Hervé Brönnimann.
5. K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17:830–847, 1988.
6. L. Devroye and T. Wagner. Nearest neighbor methods in discrimination. In North-Holland, editor, *Handbook of Statistics*, volume 2. P.R. Krishnaiah, L.N. Kanal, 1982.
7. D. P. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM J. Comput.*, 5:181–186, 1976.
8. R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.
9. R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley-Interscience, New York, 1973.
10. M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The QBIC system. *Computer*, 28:23–32, 1995.
11. P. Frankl and H. Maehara. The Johnson-Lindenstrauss lemma and the sphericity of some graphs. *Journal of Combinatorial Theory B*, 44:355–362, 1988.
12. A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Press, 1991.
13. L. Heinrich-Litan. PhD thesis, FU Berlin, in preparation.
14. P. Indyk. Dimensionality reduction techniques for proximity problems. In *Proceedings, ACM Symposium on Data Structures and Algorithms, SODA 2000*, pages 371–378, 2000.
15. P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, page to appear, 1998.
16. W. Johnson and G. Schechtman. Embedding l_p^m into l_1^n . *Acta Mathematica*, 149:71–85, 1985.
17. D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12:28–35, 1983.
18. R. Klein. *Algorithmische Geometrie*. Addison-Wesley, Bonn, 1997.
19. J. Kleinberg. Two algorithms for nearest-neighbor search in high dimension. In *Proc. 29th Annu. ACM Sympos. Theory Comput.*, pages 599–608, 1997.
20. D. Knuth. *The art of computer programming*. Addison-Wesley, 1973.
21. E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, page to appear, 1998.
22. N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.
23. S. Meiser. Point location in arrangements of hyperplanes. *Inform. Comput.*, 106:286–303, 1993.
24. M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge MA, 1969.
25. A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, UK, 1992.
26. A. W. M. Smeulders and R. J. (eds.). Image databases and multi-media search. In *Proceedings of the First International Workshop, IDB-MMS '96*, Amsterdam, 1996. Amsterdam University Press.
27. C. J. van Rijsbergen. *Information Retrieval*. Butterworths, 1979.

Explicit and Implicit Enforcing – Randomized Optimization

Bernd Gärtner and Emo Welzl

Theoretical Computer Science, ETH Zürich, CH-8092 Zürich, Switzerland
`{gaertner,emo}@inf.ethz.ch`

1 Introduction

This tutorial is aiming at some randomized approaches for geometric optimization that have been developed over the last ten to fifteen years. The methods are simple, sometimes even the analysis is. The simplicity of the methods entails that they use very little of the underlying structure of the problems to tackle, and, as a consequence, they are applicable to a whole range of problems. Most prominently, they have led to new bounds for combinatorial solutions to linear programming. But with little adaption necessary – basically few problem specific primitive operations have to be supplied – the methods can be used for computing smallest enclosing balls of point sets, smallest volume enclosing ellipsoids, the distance between polytopes (given either as convex hulls or as intersection of halfspaces), and for several other problems. If the dimension of the input instance is constant, then several linear time methods are available, and some of them even behave ‘well’ when the dimension grows; at least, no better provable behavior is known. Many of the methods can be interpreted as variants of the simplex algorithm, with appropriately chosen randomized pivot rules.

After the good news, the even better news. It seems that there is ample space for improvement. The fact that the methods are so general is nice, but we would be more than happy to use more of the structure, if we only knew how. The ultimate goal is to derive a polynomial combinatorial linear programming algorithm.

Disclaimer. We are not aiming at the most succinct presentation of the most efficient procedures (let it be theoretically or practically). Instead, we take a stroll around the topic with emphasis on the methods, which we believe to be of interest in their own right. And we enjoy how binomial coefficients and cycle numbers enter the picture. Usefulness is definitely not our primary concern!

Now that we got rid of the utilitarian fanatic, let us disclose that *some* of the ideas discussed here have found their way into efficient implementations (see, e.g., [6]).

For the rest of this section we briefly discuss two processes similar in structure to what we will later use for computing smallest enclosing balls, the problem of reference for this presentation. This algorithmic problem was posed by James Joseph Sylvester [18] in 1857. In Sect. 2 we settle basic terminology and properties

needed for the algorithmic treatment of that problem, and we analyze a first process that computes a smallest enclosing ball of a finite point set. Next, in Sect. 3, we present the first two efficient procedures: One is linear in n , the number of points, but has exponential behavior in terms of the dimension d ; the other one is exponential in n , but it is faster than the first one when the dimension is large ($d \approx n$). Assuming a fast solution for $n = 2(d + 1)$, Sect. 4 presents a subexponential process. Discussion and bibliographical remarks conclude the paper.

We are interested in the processes, emphasizing their common and their distinguished features. We omit a presentation of the best known method for solving the problem when $n \approx d$, which is quite technical, as it stands now. The best known bounds are summarized in the discussion.

Minimum Finding Procedures. Let us start with two simple procedures for computing the minimum of a set of real numbers. These procedures may seem somewhat stupid, and, in fact, they were, if we were really interested in computing minima. However, we are interested in the processes they define, and in their analyses, not in the output they produce. They already provide schemes which are able to solve more difficult problems. Here is a first try. We assume that we are given a set A of n real numbers. The procedure `AlwaysCheckMin` performs

	function <code>AlwaysCheckMin</code> (A)
	if $A = \emptyset$ then
	$x \leftarrow \infty$;
	else
$A \subseteq \mathbf{R}$, finite.	$a \leftarrow_{\text{random}} A$;
PRECONDITION:	$x \leftarrow \text{AlwaysCheckMin}(A \setminus \{a\})$;
None.	if $a < x$ then
POSTCONDITION:	$x \leftarrow a$;
$x = \min A$.	for all $b \in A$ do
	if $b < x$ then
	output "Something wrong!";
	return x ;

the normal scan through A in random order, always substituting a current minimum if a smaller number is encountered. However, whenever a new minimum is encountered, we check it against all previous numbers (not trusting the transitivity of the \leq -relation¹). We use here and further on the statement ' $s \leftarrow_{\text{random}} S$;

¹ Perhaps its not so ridiculous at all. Just imagine the numbers are given implicitly, as x -coordinates of intersections of lines, say. Then numerical problems in calculations and comparisons may quite easily lead you into contradictions. But, again, that's not the point here!

for assigning to s an element uniformly at random from set S . The expected number, $t(n)$, of comparisons (‘ $a < x$ ’ or ‘ $b < x$ ’) of the procedure when A has cardinality n obeys the recursion

AlwaysCheckMin-recursion, comparisons

$$t(n) = \begin{cases} 0, & \text{if } n = 0, \\ t(n-1) + 1 + \frac{1}{n} \cdot n, & \text{if } n \geq 1, \end{cases}$$

since the randomly chosen element a is the minimum of A with probability² $\frac{1}{n}$. Hence, $t(n) = 2n$. If some of the numbers are equal, the expected number of comparisons decreases.

Here is a second procedure, called **TryAndTryMin**, that computes the minimum of a nonempty set A of real numbers. Note that the function invokes a

$A \subseteq \mathbf{R}$, finite. PRECONDITION: $A \neq \emptyset$. POSTCONDITION: $x = \min A$.	<pre> function TryAndTryMin(A) $a \leftarrow_{\text{random}} A$; if $A = \{a\}$ then $x \leftarrow a$; else $x \leftarrow \text{TryAndTryMin}(A \setminus \{a\})$; if $a < x$ then $x \leftarrow \text{TryAndTryMin}(A)$; return x; </pre>
---	--

recursive call with the same arguments – something that is prohibited in a deterministic procedure that we wish to eventually terminate. The strange feature of the procedure is, of course, that it ignores the fact

$$a < \min(A \setminus \{a\}) \implies a = \min A .$$

It will be our task to find corresponding facts in some of the higher-dimensional counterparts, when they are not as self-evident as here.

For $n \in \mathbf{N}$, the expected number, $t(n)$, of comparisons of the procedure when applied to a set of n elements satisfies the recursion

TryAndTryMin-recursion, comparisons

$$t(n) = \begin{cases} 0, & \text{if } n = 1, \\ t(n-1) + 1 + \frac{1}{n} \cdot t(n), & \text{otherwise,} \end{cases}$$

since the second recursive call happens iff $a = \min A$, which happens with probability $\frac{1}{n}$. For $n > 1$, this can be rewritten as (by moving the $t(n)$ -term from the right to the left side, and dividing by $(n-1)$)

² If not all numbers are distinct (i.e. A is a multiset), the probability is *at most* $\frac{1}{n}$.

$$\frac{t(n)}{n} = \frac{t(n-1)}{n-1} + \frac{1}{n-1} = \frac{1}{n-1} + \frac{1}{n-2} + \cdots + \frac{1}{1} + \frac{t(1)}{1}$$

which yields

$$\frac{t(n)}{n} = H_{n-1} \implies t(n) = nH_{n-1},$$

where H_m , $m \in \mathbf{N}_0$, is the m th *Harmonic number* $H_m := \sum_{i=1}^m \frac{1}{i}$; $\ln(m+1) \leq H_m \leq 1 + \ln m$ for $m \in \mathbf{N}$.

If A is a multiset, the recursion still holds, but now with inequality (the second recursive call happens iff a is the unique minimum in A); the expected number of steps can only decrease.

Notation. \mathbf{N} denotes the positive integers, \mathbf{N}_0 the nonnegative integers, \mathbf{R} the real numbers, \mathbf{R}^+ the positive reals, and \mathbf{R}_0^+ the nonnegative reals.

$k \in \mathbf{N}_0$; A a finite set. $\#A$ denotes the cardinality of A , $\binom{A}{k}$ the family of k -element subsets of A , and 2^A the family of all subsets of A .

$n, k \in \mathbf{N}_0$. $n^{\underline{k}}$ denotes the *falling power* $\prod_{i=0}^{k-1} (n-i)$, $\binom{n}{k}$ the *binomial coefficient* $\frac{n^{\underline{k}}}{k!}$, and $\left[\begin{smallmatrix} n \\ k \end{smallmatrix} \right]$ the *cycle number* (Stirling number of the first kind). It counts the number of permutations of n elements with k cycles. Equivalently, it is defined by $\left[\begin{smallmatrix} 0 \\ 0 \end{smallmatrix} \right] = 1$, and for $n, k \in \mathbf{N}$, $\left[\begin{smallmatrix} n \\ 0 \end{smallmatrix} \right] = \left[\begin{smallmatrix} 0 \\ k \end{smallmatrix} \right] = 0$ and

$$\left[\begin{smallmatrix} n \\ k \end{smallmatrix} \right] = (n-1) \left[\begin{smallmatrix} n-1 \\ k \end{smallmatrix} \right] + \left[\begin{smallmatrix} n-1 \\ k-1 \end{smallmatrix} \right].$$

For properties of falling powers, binomial coefficients and cycle numbers the reader is invited to consult the exposition [11].

$d \in \mathbf{N}$; $x \in \mathbf{R}^d$. $\|x\|$ denotes the length of vector x (2-norm).

$A \subseteq \mathbf{R}$. We use $\max A$, $\sup A$, $\min A$, and $\inf A$ to denote the maximum, supremum, minimum, and infimum, resp., of A . As usual, $\min \emptyset = \inf \emptyset = \infty$ and $\max \emptyset = \sup \emptyset = -\infty$.

Convention. Throughout this text, d is a positive integer (the dimension of the ambient space), and $B, C, E, F, P, S \subseteq \mathbf{R}^d$ are *finite* without further explicit mention.

2 Smallest Enclosing Ball

Here is our reference problem for this presentation. We are given a set P of points in \mathbf{R}^d and we are to compute a closed ball of smallest radius containing P . The nice feature of this optimization problem (as opposed to linear programming, say) is that a solution always exists, and it is always unique.

$z \in \mathbf{R}^d$; $r \in \mathbf{R}$. We use $b_{z,r}$ for the closed ball with center z and radius r , $b_{z,r} := \{p \in \mathbf{R}^d \mid \|p-z\| \leq r\}$. Given such a ball b , we let ∂b denote the boundary of b . If $b = b_{z,r} \neq \emptyset$, we let $z_b := z$ and $r_b := r$; $r_\emptyset := -\infty$.

Note that $b_{z,r}$ degenerates to a point iff $r = 0$, and to the empty set iff $r < 0$.

Definition 1 (smallest enclosing radius). $P \subseteq \mathbf{R}^d$.

$$\rho P := \inf_{\text{closed ball } b \supseteq P} r_b .$$

In particular, $\rho\emptyset = -\infty$; $\rho P = 0$ iff $\#P = 1$; and $\rho P > 0$, otherwise.

We omit the proof for the geometric lemma below. The reader may wish to read (2-4) as ‘axioms’ that are necessary for the later conclusions. In this way the generality of the results becomes evident.

Lemma 1. $P \subseteq \mathbf{R}^d$.

- (1) *There is a unique closed ball of radius ρP that contains P .*
- (2) *If $P' \subseteq P$ then $\rho P' \leq \rho P$.*
- (3) *Let $P' \subseteq P$. Then $\rho P' = \rho P$ iff $\rho(P' \cup \{p\}) = \rho P'$ for all $p \in P \setminus P'$.*
- (4) *If $\rho P' \neq \rho P$ for all proper subsets P' of P , then $\#P \leq d + 1$.*

While (1) and (4) require some geometric reasoning, (2) is immediate from the definition of ρ , and (3) is immediate from (1) and (2). In order to properly appreciate³ (3), note that it is possible⁴ for $P' \subseteq P$ that $\rho P' \neq \rho P$ while $\rho P = \rho(P \setminus \{p\})$ for all $p \in P \setminus P'$.

Definition 2 (smallest enclosing ball, basis, basis of, extreme).
 $P \subseteq \mathbf{R}^d$.

- (1) *$\min BP$ denotes the ball of radius ρP covering P .*
- (2) *P is called a basis if $\min BP' \neq \min BP$ for all proper subsets P' of P .*
- (3) *A basis $B \subseteq P$ is called basis of P if $\min BB = \min BP$.*
- (4) *A point $p \in P$ is called extreme in P if $\min B(P \setminus \{p\}) \neq \min BP$.*

As it can be easily seen, there is always a basis of P . But, in general, the basis of a set P is not unique⁵. (Take, for example, the four vertices of a square. Observe also, that this set has no extreme points at all.) Moreover, note that

$$\min BP' = \min BP \iff \rho P' = \rho P, \quad \text{if } P' \subseteq P.$$

Lemma 2. $P \subseteq \mathbf{R}^d$.

- (1) *A point $p \in P$ is extreme in P iff p is contained in every basis of P .*
- (2) *The number of extreme elements in P is at most $d + 1$.*

Proof. (1) Let p be an extreme point in P and let B be a basis of P which does not contain p . Then $\rho B \leq \rho(P \setminus \{p\}) < \rho P = \rho B$; a contradiction.

On the other hand, if $\min BP = \min B(P \setminus \{p\})$ – that is, p is not extreme – then any basis of $P \setminus \{p\}$ is a basis of P that does not contain p .

- (2) Follows from (1) and Lemma 1(4). □

³ Another exercise for appreciation is to define σP as the smallest volume of a simplex containing P , and then to see how a statement analogous to Lemma 1(3) fails.

⁴ This can, however, be excluded by certain general position assumptions.

⁵ It is unique under appropriate general position assumptions.

Basic (Primitive) Operations. Throughout the presentation we will measure algorithms in terms of some basic operations. The fact that we do not elaborate much on them does not imply that they are trivial to solve efficiently – definitely not so, when it comes to actual implementations. Two such basic operations for points in \mathbf{R}^d are:

- *(d, k) -Basis computation* ‘ $\text{basis}_k^d S$ ’, for $d, k \in \mathbf{N}$: Given a set $S \subseteq \mathbf{R}^d$ with $\#S \leq k$, it returns a basis of S .
- *Inclusion predicate* ‘ $p \notin \textcircled{B}$ ’: Given a *basis* B and a point p , it decides whether p is not in $\min BB$.

If an algorithm delivers a basis B of the given point set P , then it is easy to determine $\min BB = \min BP$: compute center and radius of the unique ball in the affine hull of B that has B on its boundary⁶.

If basis_k^d were available for all k , then, of course, there is little to do. However, we will assume that this operation is at disposal only for certain k ’s dependent on d , ($k = d + 1$ and $k = 2(d + 1)$). Again, we do not claim that these are easy to solve – in fact, they are at the core of the problem, and no polynomial time solutions are known.

What is an obvious bound for obtaining a basis of a set P of $n \geq d + 1$ points in \mathbf{R}^d ? We can go through all $(d + 1)$ -element subsets, apply basis_{d+1}^d to each of them, and check the obtained basis against all remaining $n - (d + 1)$ points with inclusion tests. This requires $\binom{n}{d+1}$ calls to basis_{d+1}^d and $\binom{n}{d+1}(n - d - 1)$ inclusion tests. Instead of checking each basis against all remaining points, we could simply search for a basis B that maximizes ρB among all bases. The reader is encouraged to verify that this will indeed deliver a basis of P .

Trial and Error. In the spirit of the procedures in the previous section, and equipped with the basic operation basis_{d+1}^d and the predicate $p \notin \textcircled{B}$, we can easily provide a first randomized procedure called **TryAndTry** for our problem. It takes a set $S \subseteq \mathbf{R}^d$ and returns a basis of S . Correctness of the procedure is obvious, provided it eventually terminates.

Let $t(n)$ denote the maximum⁷ expected number of ‘ $p \notin \textcircled{B}$ ’-tests performed by the algorithm for an n -point set in \mathbf{R}^d . Then

$$\begin{array}{c} \text{TryAndTry-recursion, inclusion tests} \\ t(n) \begin{cases} = 0, & \text{if } n \leq d + 1, \\ \leq t(n - 1) + 1 + \frac{d+1}{n} \cdot t(n), & \text{otherwise,} \end{cases} \end{array}$$

since there are at most $d + 1$ extreme points in S whose removal entails the second recursive call. Division of the inequality by $(n - 1)^{\frac{d+1}{d}}$ and carrying the

⁶ In time $O(d^3)$. Similarly, the inclusion tests can be implemented efficiently: If center and radius of $\min BB$ have been precomputed, in time $O(d)$; if not, in $O(d^3)$.

⁷ By ‘maximum expected’ we mean the following: Let $T(S)$ be the expected number of tests for $S \subseteq \mathbf{R}^d$, and $t(n) := \sup_{\#S=n} T(S)$.

	function TryAndTry(S)
	if $\#S \leq d + 1$ then
	$B \leftarrow \text{basis}_{d+1}^d S$;
	else
	$p \leftarrow_{\text{random}} S$;
	$B \leftarrow \text{TryAndTry}(S \setminus \{p\})$;
	if $p \notin \textcircled{B}$ then
	$B \leftarrow \text{TryAndTry}(S)$;
	return B ;

$t(n)$ -term to the left side gives⁸

$$\begin{aligned}
 \frac{t(n)}{n^{d+1}} &\leq \frac{t(n-1)}{(n-1)^{d+1}} + \frac{1}{(n-1)^{d+1}} \\
 &\leq \frac{1}{(n-1)^{d+1}} + \frac{1}{(n-2)^{d+1}} + \cdots + \frac{1}{(d+1)^{d+1}} + \frac{t(d+1)}{(d+1)^{d+1}} \\
 &= \frac{1}{d} \left(\frac{1}{d^d} - \frac{1}{(n-1)^d} \right) = \frac{1}{(d+1)!} \left(1 + \frac{1}{d} \right) - \frac{1}{d(n-1)^d}
 \end{aligned}$$

for $n > d + 1$. The recursion for the maximum expected number, $b(n)$, of basis computations is

$$\begin{aligned}
 &\text{TryAndTry-recursion, basis}_{d+1}^d\text{-computations} \\
 b(n) &\begin{cases} = 1, & \text{if } n \leq d + 1, \\ \leq b(n-1) + \frac{d+1}{n} \cdot b(n), & \text{otherwise.} \end{cases}
 \end{aligned}$$

For $n \geq d + 1$, this matches exactly the recursion

$$\binom{n}{d+1} = \begin{cases} 1, & \text{if } n = d + 1 \\ \binom{n-1}{d+1} + \frac{d+1}{n} \binom{n}{d+1}, & \text{if } n > d + 1, \end{cases}$$

for binomial coefficients.

Theorem 1. $n \in \mathbf{N}_0$; $P \subseteq \mathbf{R}^d$, $\#P = n \geq d + 1$.

Procedure TryAndTry computes a basis of P with an expected number of at most

$$\binom{n}{d+1} \left(1 + \frac{1}{d} \right) - \frac{n}{d} \quad \text{inclusion-tests}$$

and with an expected number of at most

$$\binom{n}{d+1} \quad \text{employments of basis}_{d+1}^d\text{-computations.}$$

⁸ For the identity for sums of reciprocals of falling powers consult [11, page 53, equation (2.53)].

3 Boundary (Explicit) Enforcing

What is wrong with TryAndTry? Recall why we thought that TryAndTryMin from the previous section was so ridiculous: It ignored the useful information that $a < \min(A \setminus \{a\})$ implies that a is already the minimum. Here things are not as immediate, but we know that $p \notin \min B(P \setminus \{p\})$ implies that p is in every basis of P . Moreover, it can be shown that

$$p \notin \min B(P \setminus \{p\}) \implies p \text{ lies on the boundary of } \min B P$$

We should transfer this useful information to the second recursive call, and we will do so by *explicitly forcing* p on the boundary. If this extra information would suffice to allow a linear time solution, then the whole procedure would be linear, for reasons similar to the fact that function AlwaysCheckMin was linear.

Procedure Explicit (to be described shortly) computes the smallest enclosing ball of a given set S , with a given subset $E \subseteq S$ of points prescribed to lie on the boundary of the ball. We have to lay the basics for description and analysis first.

Definition 3 (prescribed boundary: smallest enclosing radius).
 $E \subseteq P \subseteq \mathbf{R}^d$.

$$\bar{\rho}(P, E) := \inf_{\text{closed ball } b \supseteq P, \partial b \supseteq E} r_b .$$

In particular, if there is no ball that covers P and has E on its boundary then $\bar{\rho}(P, E) = \infty$.

Lemma 3. $E \subseteq P \subseteq \mathbf{R}^d$.

(1) If $\bar{\rho}(P, E) \neq \infty$, there is a unique ball of radius $\bar{\rho}(P, E)$ that contains P and has E on its boundary.

(2) If $E' \subseteq E$ and $E' \subseteq P' \subseteq P$ then $\bar{\rho}(P', E') \leq \bar{\rho}(P, E)$.

(3) Let $E' \subseteq E$ and $E' \subseteq P' \subseteq P$. Then $\bar{\rho}(P', E') = \bar{\rho}(P, E)$ iff

$$\bar{\rho}(P' \cup \{p\}, E') = \bar{\rho}(P', E') \text{ for all } p \in P \setminus P',$$

and

$$\bar{\rho}(P' \cup \{p\}, E' \cup \{p\}) = \bar{\rho}(P', E') \text{ for all } p \in E \setminus E'.$$

(4) If $\bar{\rho}(P', E') \neq \bar{\rho}(P, E) < \infty$ for all $(P', E') \neq (P, E)$ with $E' \subseteq E$ and $E' \subseteq P' \subseteq P$, then $\#P' \leq d + 1$.

Similar to Lemma 1, (2) is obvious from the definition, and (3) is a consequence of (1) and (2).

Definition 4 (prescribed boundary: smallest enclosing ball, extreme).
 $E \subset P \subseteq \mathbf{R}^d$.

(1) If $\bar{\rho}(P, E) < \infty$, $\min \bar{B}(P, E)$ denotes the ball of radius $\bar{\rho}(P, E)$ that contains P and has E on its boundary. If $\bar{\rho}(P, E) = \infty$, we set $\min \bar{B}(P, E) = \mathbf{R}^d$ (indicating infeasibility).

(2) A point $p \in P \setminus E$ is called *extreme* in (P, E) if $\min \bar{B}(P \setminus \{p\}, E) \neq \min \bar{B}(P, E)$.

Lemma 4. $E \subseteq P \subseteq \mathbf{R}^d$.

(1) If $p \in P \setminus E$ is extreme in (P, E) , then

$$\min \bar{\mathbf{B}}(P, E) = \min \bar{\mathbf{B}}(P, E \cup \{p\})$$

and

$$\min \bar{\mathbf{B}}(P, E) = \mathbf{R}^d \quad \text{or} \quad p \text{ is affinely independent from the points in } E.$$

(2) If E is affinely independent and $\min \bar{\mathbf{B}}(P, E) \neq \mathbf{R}^d$, then the number of extreme points in (P, E) is at most $d + 1 - \#E$.

It is instructive to display the implication of assertion (1) of the lemma in the following form.

$$\min \bar{\mathbf{B}}(P, E) = \begin{cases} \min \bar{\mathbf{B}}(P \setminus \{p\}, E), & \text{if } p \in \min \bar{\mathbf{B}}(P \setminus \{p\}, E), \\ \min \bar{\mathbf{B}}(P, E \cup \{p\}), & \text{if } p \notin \min \bar{\mathbf{B}}(P \setminus \{p\}, E), \end{cases} \quad (1)$$

for all $p \in P \setminus E$.

We will also need a new predicate⁹

- (Boundary) inclusion predicate ‘ $p \notin \textcircled{F}$ ’: Given a set F of affinely independent points and a point p , it decides whether p is not in $\min \bar{\mathbf{B}}(F, F)$.

Forcing Points on the Boundary. Here is the procedure **Explicit** that computes the smallest enclosing ball of a set S of points, with a set $E \subseteq S$ of points that are prescribed to lie on the boundary of the ball.

$E, S \subseteq \mathbf{R}^d$. PRECONDITION: $E \subseteq S$; E affinely independent; $\min \bar{\mathbf{B}}(S, E) \neq \mathbf{R}^d$. POSTCONDITION: $E \subseteq F \subseteq S$; F affinely independent; $\min \bar{\mathbf{B}}(F, F) = \min \bar{\mathbf{B}}(S, E)$.	function Explicit (S, E) if $S = E$ then $F \leftarrow E$; else $p \leftarrow_{\text{random}} S \setminus E$; $F \leftarrow \text{Explicit}(S \setminus \{p\}, E)$; if $p \notin \textcircled{F}$ then $F \leftarrow \text{Explicit}(S, E \cup \{p\})$; return F ;
--	--

From the precondition and (1) it follows that the precondition is satisfied for the recursive calls and that the function returns a set $F \supseteq E$ of affinely independent points with $\min \bar{\mathbf{B}}(F, F) = \min \bar{\mathbf{B}}(S, E)$. Our original problem is solved by the call **Explicit**(P, \emptyset) – the arguments obviously satisfy the precondition.

⁹ The implementation of this predicate is in fact identical to that of the inclusion predicate for bases: We compute the unique ball in the affine hull of F that has F on its boundary. Its center and radius are also center and radius of $\min \bar{\mathbf{B}}(F, F)$. Now the predicate can be evaluated by a simple distance calculation.

For $k, n \in \mathbf{N}_0$, $t_k(n)$ counts the maximum expected number of inclusion tests when $\#(S \setminus E) = n$ and $d+1 - \#E = k$. The parameter k expresses the degrees of freedom¹⁰. Since there are at most k extreme points in $S \setminus E$, we get the following recursion. Note that $k = 0$ means that $\#E = d+1$. Hence, any other point is affinely dependent on E , and so a recursive call with parameters $(S, E \cup \{p\})$ would lead to a contradiction. Therefore, $t_0(n) = t_0(n-1) + 1$ for $n \in \mathbf{N}$.

Explicit-recursion, inclusion tests

$$t_k(n) \begin{cases} = 0, & \text{if } n = 0, \\ = t_0(n-1) + 1, & \text{if } k = 0 \text{ and } n > 0, \\ \leq t_k(n-1) + 1 + \frac{k}{n} \cdot t_{k-1}(n-1), & \text{if } n \geq 1 \text{ and } k \geq 1. \end{cases}$$

From $t_0(n) = n$ we get $t_1(n) \leq 2n$, and that yields $t_2(n) \leq 5n$, etc. Hence, we might guess that there is a bound of the form $t_k(n) \leq c_k n$ for constants c_k , $k \in \mathbf{N}_0$. These constants satisfy

$$c_k = \begin{cases} 1, & \text{if } k = 0, \\ 1 + kc_{k-1}, & \text{if } k \geq 1. \end{cases}$$

Division by $k!$ lets us rewrite this in the form

$$\frac{c_k}{k!} = \frac{1}{k!} + \frac{c_{k-1}}{(k-1)!} = \frac{1}{k!} + \frac{1}{(k-1)!} + \cdots + \frac{1}{1!} + \frac{c_0}{0!},$$

and we conclude that

$$c_k = k! \sum_{i=0}^k \frac{1}{i!} = \begin{cases} 1, & \text{if } k = 0, \\ \lfloor e k! \rfloor, & \text{if } k \geq 1. \end{cases}$$

The inequality $t_k(n) \leq c_k n$ for all $k, n \in \mathbf{N}_0$ is now easy to prove by induction.

Let us also estimate the maximum expected number $b_k(n)$ of assignments ' $F \leftarrow E$;' (when $S = E$). We call these the *base cases*¹¹, and we obtain the recursion

Explicit-recursion, base cases

$$b_k(n) \begin{cases} = 1, & \text{if } n = 0 \text{ or } k = 0, \\ \leq b_k(n-1) + \frac{k}{n} \cdot b_{k-1}(n-1), & \text{if } n \geq 1 \text{ and } k \geq 1. \end{cases}$$

We multiply both sides by $\frac{n!}{k!}$ to obtain

$$\begin{aligned} \overbrace{\frac{n! b_k(n)}{k!}}^{B_{k+1}(n+1)} &\leq n \cdot \underbrace{\frac{(n-1)! b_k(n-1)}{k!}}_{=B_{k+1}(n)} + \underbrace{\frac{(n-1)! b_{k-1}(n-1)}{(k-1)!}}_{=B_k(n)} \end{aligned}$$

¹⁰ And note: Less freedom is better!

¹¹ If you think about an actual implementation, then this is a relevant quantity, since it is (perhaps) advisable to precompute center and radius of $\min \bar{\mathbf{B}}(F, F)$ in $O(d^3)$, and return F with this information, so that further inclusion tests can be performed more efficiently in $O(d)$.

which reminds us of the recursion for the cycle numbers. In fact, we also have $B_1(n+1) = n! = \left[\begin{smallmatrix} n+1 \\ 1 \end{smallmatrix} \right]$ for $n \in \mathbf{N}_0$, but for $k \in \mathbf{N}$, $B_{k+1}(1) = \frac{1}{k!} \neq \left[\begin{smallmatrix} 1 \\ k+1 \end{smallmatrix} \right] = 0$.

It can be shown that if ‘ \leq ’ is replaced by ‘ $=$ ’ then the solution to the recurrence for $b_k(n)$ equals

$$\begin{aligned} \frac{1}{n!} \sum_{i=0}^k \binom{k}{i} i! \left[\begin{smallmatrix} n+1 \\ i+1 \end{smallmatrix} \right] &\leq \frac{1}{n!} \sum_{i=0}^k \binom{k}{i} i! \left(\frac{n!}{i!} (H_n)^i \right) \quad \text{see Appendix, Lemma 10} \\ &= \sum_{i=0}^k \binom{k}{i} (H_n)^i = (1 + H_n)^k \end{aligned}$$

Unless n is small compared to d , the number of base cases is much smaller than the number of inclusion tests, so precomputation pays off.

Theorem 2. $n \in \mathbf{N}_0$; $P \subseteq \mathbf{R}^d$, $\#P = n$.

A call $\text{Explicit}(P, \emptyset)$ computes a subset F of P with $\min \bar{\mathbf{B}}(F, F) = \min \mathbf{B}P$ with an expected number of at most

$$\lfloor e(d+1)! \rfloor n \quad \text{inclusion tests}$$

and an expected number of at most

$$\frac{1}{n!} \sum_{i=0}^{d+1} \binom{d+1}{i} \frac{1}{i!} \left[\begin{smallmatrix} n+1 \\ i+1 \end{smallmatrix} \right] \leq (1 + H_n)^{d+1} \quad \text{base cases.}$$

To Force or not to Force. After choosing point p , procedure Explicit has two options, either (i) to omit the point p from consideration, or (ii) to assume that p is extreme and to force it on the boundary. And it decides to bet on (i) first, and look at (ii) only if (i) failed. This is a reasonable thing to do, since – most likely – a random point will not be extreme. But this is not true if the number of points is small compared to the dimension (or, more precisely, if the number of points available in $S \setminus E$ is small compared to the number $d+1 - \#E$ of empty slots available). Actually, as you think about it, the recursion for $t_k(n)$ becomes somewhat strange when $n < k$: We account a ‘probability’ larger than 1 for proceeding to the second call (the recursion is still correct, but obviously not tight in that case).

So we design a new procedure ForceOrNot ¹² which will perform favorably for n small compared to k . This procedure chooses an arbitrary point p in $S \setminus E$, and then tosses a coin to decide whether option (i) or (ii) as described above is pursued.

New features appear. First, if we force *some* point on the boundary we have no guarantee that a ball with the prescribed properties exists. We circumvent this problem by requiring that the points in S are affinely independent. Hence, there exists a ball with S on its boundary and so $\min \bar{\mathbf{B}}(S', E') \neq \mathbf{R}^d$ for all

¹² ‘ $s \leftarrow_{\text{some}} S$,’ assigns to s an *arbitrary* element in S .

```

function ForceOrNot( $S, E$ )
if  $S = E$  then
     $F \leftarrow E$ ;
else
     $p \leftarrow_{\text{some}} S \setminus E$ ;
     $x \leftarrow_{\text{random}} \{0, 1\}$ ;
    if  $x = 1$  then
         $F \leftarrow \text{ForceOrNot}(S \setminus \{p\}, E)$ ;
        if  $p \notin \textcircled{F}$  then
             $F \leftarrow \text{ForceOrNot}(S, E \cup \{p\})$ ;
        else
             $F \leftarrow \text{ForceOrNot}(S, E \cup \{p\})$ ;
        if  $p$  loose in  $F$  then
             $F \leftarrow \text{ForceOrNot}(S \setminus \{p\}, E)$ ;
return  $F$ ;

```

$E' \subseteq S' \subseteq S$. This implies that $\#S \leq d + 1$, which is not so much of a problem, since we wanted a procedure for small sets to begin with. Moreover, we make the following

General Position Assumption. (2)

For all $\emptyset \neq F \subseteq S$, no point in $S \setminus F$ lies on the boundary of $\min\overline{B}(F, F)$.

Both assumptions, affine independence and general position, can be attained via symbolic perturbation techniques as long as the number of points does not exceed¹³ $d + 1$.

Second, how do we know now that a call $\text{ForceOrNot}(S, E \cup \{p\})$ delivers a description of the smallest ball enclosing S and with E on its boundary? We were not concerned about that before in procedure **Explicit**, since there we made such a call only if we knew that p has to lie on the boundary of such a ball.

Definition 5 (loose). $F \subseteq \mathbf{R}^d$.

A point $p \in F$ is called loose in F if $\min\overline{B}(F, F \setminus \{p\}) \neq \min\overline{B}(F, F)$.

The following lemmas show that we can check $\min\overline{B}(F, F) = \min\overline{B}(P, E)$ by local inclusion and looseness tests.

Lemma 5. $E \subseteq F \subseteq P \subseteq \mathbf{R}^d$, P affinely independent and in general position according to (2).

$\min\overline{B}(F, F) = \min\overline{B}(P, E)$ iff

$$\min\overline{B}(F \cup \{p\}, F) = \min\overline{B}(F, F) \text{ for all } p \in P \setminus F \quad (*)$$

¹³ And if the number n of points exceeds $d + 1$, then embed \mathbf{R}^d in \mathbf{R}^{n-1} , and perturb!

and

$$\min\bar{B}(F, F \setminus \{p\}) = \min\bar{B}(F, F) \text{ for all } p \in F \setminus E. \quad (**)$$

While (*) can be checked by the boundary inclusion predicate, we need a new predicate¹⁴ corresponding to condition (**).

- *Looseness predicate* ‘ p loose in F ’: Given a set F of affinely independent points and a point p , it decides whether p is loose in F .

Now that the correctness of the procedure is established, let us consider efficiency. For a point p in $S \setminus E$ we have that $\min\bar{B}(S, E) = \min\bar{B}(S, E \cup \{p\})$ or $\min\bar{B}(S, E) = \min\bar{B}(S \setminus \{p\}, E)$. Our general position assumption excludes that both are true. Hence, if we toss a coin to guess which one is true, then we are mistaken with probability $\frac{1}{2}$.

Let $t(n)$ denote the expected number of inclusion and looseness tests when $\#(S \setminus E) = n$. Then

$$\begin{array}{l} \text{ForceOrNot-recursion, inclusion and looseness tests} \\ t(n) = \begin{cases} 0, & \text{if } n = 0, \\ t(n-1) + 1 + \frac{1}{2} \cdot t(n-1), & \text{otherwise.} \end{cases} \end{array}$$

That is,

$$t(n) = 1 + \frac{3}{2} \cdot t(n-1) = 1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \cdots + \left(\frac{3}{2}\right)^{n-1} + \left(\frac{3}{2}\right)^n t(0)$$

and so $t(n) = 2 \left(\left(\frac{3}{2}\right)^n - 1 \right)$. For example, $t(10) \approx 113$, $t(20) \approx 6,684$, $t(30) \approx 383,500$, and $t(40) \approx 22,114,662$. Here is the recursion for the expected number of base cases.

$$\begin{array}{l} \text{ForceOrNot-recursion, base cases} \\ b(n) = \begin{cases} 1, & \text{if } n = 0, \\ b(n-1) + \frac{1}{2} \cdot b(n-1), & \text{otherwise,} \end{cases} \end{array}$$

which solves to $\left(\frac{3}{2}\right)^n$. We learnt that on the average roughly two inclusion tests are performed for each F appearing in the procedure.

Theorem 3. $n \in \mathbb{N}_0$; $P \subseteq \mathbb{R}^d$, $\#P = n$, P affinely independent and in general position according to (2).

A call $\text{ForceOrNot}(P, \emptyset)$ computes a subset F of P with $\min\bar{B}(F, F) = \min B P$ with an expected number of

$$2 \left(\left(\frac{3}{2}\right)^n - 1 \right) \quad \text{inclusion and looseness tests}$$

and an expected number of

$$\left(\frac{3}{2}\right)^n \quad \text{base cases.}$$

¹⁴ An implementation of this predicate has to decide whether p is in $\min\bar{B}(F \setminus \{p\}, F \setminus \{p\})$, and, if so, whether the radius of this ball is smaller than that of $\min\bar{B}(F, F)$. All of this can be done in time $O(d^3)$.

4 Implicit Enforcing

Throughout this section we will use δ short for $d + 1$.

The procedure of the section, **Implicit**, has a very similar look and feel as **Explicit** and **TryAndTry**. Besides the set S under consideration, it has as second argument some basis $B \subseteq S$ (not necessarily a basis of S), which has no influence on the postcondition. However, it carries the essential information responsible for the efficiency of the algorithm.

$B, S \subseteq \mathbf{R}^d$. PRECONDITION: $B \subseteq S$; B basis. POSTCONDITION: C basis of S .	<pre> function Implicit(S, B) if $\#S \leq 2\delta$ then $C \leftarrow \text{basis}_{2\delta}^d S$; else $p \leftarrow_{\text{random}} S \setminus B$; $C \leftarrow \text{Implicit}(S \setminus \{p\}, B)$; if $p \notin \odot$ then $C \leftarrow \text{Implicit}(S, \text{pivot}(C, p))$; return C; </pre>
--	--

A new basic operation is required¹⁵

- *Pivot step* ‘ $\text{pivot}(B, p)$ ’: Given a basis B and a point $p \notin \min BB$, it returns a basis $B' \subseteq B \cup \{p\}$ with $\rho B' > \rho B$.

Correctness of the procedure is obvious, provided it terminates. The fact that the procedure always terminates follows from the fact, that in the first recursive call the size of the first argument decreases by 1, while in the second call we know that $\rho \text{pivot}(C, p) > \rho B$, and there are only a finite number of radii attained by bases.

As we will see, the efficiency of **Implicit** stems from the fact that the second argument, the basis B , *implicitly enforces* some of the extreme points by excluding them as possible random choices for p in this *and* all recursive calls entailed by the call with (S, B) . In fact, we will see that in the second call (if it happens at all) the number of ‘available’ (i.e. not enforced) extreme points roughly halves on the average. Here are the crucial notions.

Definition 6 (enforced, hidden dimension). $B \subseteq P \subseteq \mathbf{R}^d$, and B basis.

- (1) A point $p \in P$ is called *enforced* in (P, B) , if p is contained in all bases $C \subseteq P$ with $\rho C \geq \rho B$.
- (2) The hidden dimension $\text{hdim}(P, B)$ of (P, B) is $\delta - i$ where i is the number of enforced elements in (P, B) .

¹⁵ It allows an $O(d^3)$ implementation, although that needs some thinking, cf. [8, 7].

In particular, if p is enforced in (P, B) , then $p \in B$.

Next we list a number of simple facts. The reader should confirm their obviousness for her- or himself, so that we have them handy in the proof of the recursion for the number of basis computations.

Lemma 6. $B \subseteq P \subseteq \mathbf{R}^d$, and B basis.

- (1) A point $p \in P$ is enforced in (P, B) iff $\rho(P \setminus \{p\}) < \rho B$.
- (2) Every enforced element in (P, B) is extreme in P . There are at most δ enforced elements in (P, B) and $\text{hdim}(P, B) \geq 0$.
- (3) If $\text{hdim}(P, B) = 0$, then B is a basis of P .
- (4) If $B \subseteq P' \subseteq P$, then every enforced element in (P, B) is also enforced in (P', B) and $\text{hdim}(P', B) \leq \text{hdim}(P, B)$.
- (5) If $B' \subseteq P$ is a basis with $\rho B' \geq \rho B$ then every enforced element in (P, B) is also enforced in (P, B') and $\text{hdim}(P, B') \leq \text{hdim}(P, B)$.

Proof. (1) (\Rightarrow) If $\rho(P \setminus \{p\}) \geq \rho B$, then any basis C of $P \setminus \{p\}$ is a basis with $\rho C \geq \rho B$ and $p \notin C$, and so p is not enforced.

(\Leftarrow) If $\rho(P \setminus \{p\}) < \rho B$ and C is any basis that does not include p then $\rho C \leq \rho(P \setminus \{p\})$. Hence, $\rho C < \rho B$. Therefore, $\rho C \geq \rho B$ implies $p \in C$.

(2-5) are immediate consequences of (1). \square

For $k, n \in \mathbf{N}_0$, let $t_k(n)$, and $b_k(n)$ denote the maximum expected number of inclusion tests, and $\text{basis}_{2\delta}^d$ -computations, respectively, for a call $\text{Implicit}(S, B)$ with $\#S = n$ and $\text{hdim}(S, B) \leq k$. The number of pivot steps is always one less than the number of basis computations. This follows easily from a proof by induction over recursive calls.

We claim the following recursions (to be proven below).

Implicit-recursion, inclusion tests

$$t_k(n) \begin{cases} = 0, & \text{if } n \leq 2\delta, \\ \leq t_k(n-1) + 1 + \frac{1}{n-\delta} \cdot \sum_{i=0}^{k-1} t_i(n), & \text{otherwise.} \end{cases}$$

In particular, $t_0(n) = n - 2\delta$. A proof by induction verifies $t_k(m + 2\delta) \leq 2^k m$ for $k, m \in \mathbf{N}_0$. However, we will see that for m small, a better bound in terms of k can be derived.

Implicit-recursion, $\text{basis}_{2\delta}^d$ -computations

$$b_k(n) \begin{cases} = 1, & \text{if } n \leq 2\delta, \\ \leq b_k(n-1) + \frac{1}{n-\delta} \cdot \sum_{i=0}^{k-1} b_i(n), & \text{otherwise.} \end{cases} \quad (3)$$

In particular, $b_0(n) = 1$.

Proof of recursion (3) (number of basis computations). Clearly, $b_k(n) = 1$ for $n \leq 2\delta$. Let $B \subseteq S \subseteq \mathbf{R}^d$, $\#S = n > 2\delta$, B a basis, $\text{hdim}(S, B) \leq k \in \mathbf{N}_0$.

(i) The first recursive call solves a problem of size $n - 1$ and hidden dimension at most k .

(ii) There are at most k elements in $S \setminus B$ which trigger a second recursive call: They all have to be extreme elements; there are at most δ extreme elements, and at least $\delta - k$ of them are enforced in (S, B) (and thus included in B).

Let us denote these elements by p_1, p_2, \dots, p_r , $r \leq k$, enumerated in a way that

$$\rho(S \setminus \{p_1\}) \leq \rho(S \setminus \{p_2\}) \leq \dots \leq \rho(S \setminus \{p_r\}) .$$

If $p = p_\ell$ is chosen for the first recursive call (which happens with probability $\frac{1}{\#(S \setminus B)} \leq \frac{1}{n - \delta}$), then we receive C , a basis of $S \setminus \{p_\ell\}$. But then,

$$\rho(S \setminus \{p_1\}) \leq \rho(S \setminus \{p_2\}) \leq \dots \leq \rho(S \setminus \{p_\ell\}) = \rho C < \rho \text{pivot}(C, p) .$$

It follows that p_1, p_2, \dots, p_ℓ are enforced in $(S, \text{pivot}(C, p))$, in addition to the previously enforced elements. That is, the hidden dimension of $(S, \text{pivot}(C, p))$ is at most $k - \ell$.

Summing up, the expected number of basis computations entailed by the second recursive call is bounded by

$$\frac{1}{n - \delta} \cdot \sum_{\ell=1}^r b_{k-\ell}(n) \leq \frac{1}{n - \delta} \cdot \sum_{i=0}^{k-1} b_i(n) .$$

□

For the analysis of $b_k(n)$ we employ a function $\mathbf{N}_0^2 \rightarrow \mathbf{R}$ defined by

$$C_k(m) = \begin{cases} 1, & \text{if } k = 0, \\ 0, & \text{if } k > 0 \text{ and } m = 0, \\ C_k(m-1) + \frac{1}{m} \cdot \sum_{i=0}^{k-1} C_i(m-1), & \text{otherwise.} \end{cases}$$

This function will allow us an estimate for the expected number of basis computations via the following two steps.

Lemma 7. $k, n \in \mathbf{N}_0$; $n \geq 2\delta$.

$$b_k(n) \leq C_k(n - 2\delta + k).$$

Lemma 8. $k, m \in \mathbf{N}_0$.

$$C_k(m) = \frac{1}{m!} \sum_{i=0}^m \binom{k-1}{i-1} \left[\frac{m+1}{i+1} \right] \leq e^{2\sqrt{kH_m}} .$$

Here we adopt the convention that, for $j \in \mathbf{N}$, $\binom{j-1}{-1} = 0$ and $\binom{-1}{-1} = 1$.

Proof by induction of Lemma 7. For $k = 0$,

$$b_0(n) = 1 \leq C_0(n - 2\delta)$$

holds as required, since $C_0(n - 2\delta) = 1$.

For $k \geq 1$ and $n = 2\delta$, the assertion claims

$$b_k(2\delta) = 1 \leq C_k(k) ,$$

which holds since $C_k(k) \geq C_k(k-1) \geq \dots \geq C_k(1) = C_k(0) + C_0(0) = 1$.

For $n > 2\delta, k > 0$

$$\begin{aligned} b_k(n) &\leq b_k(n-1) + \frac{1}{n-\delta} \sum_{i=0}^{k-1} b_i(n) \\ &\leq C_k(n-1-2\delta+k) + \frac{1}{n-\delta} \sum_{i=0}^{k-1} C_i(n-2\delta+i) \\ &\leq C_k(n-2\delta+k-1) + \frac{1}{n-2\delta+k} \sum_{i=0}^{k-1} C_i(n-2\delta+k-1) \\ &= C_k(n-2\delta+k) . \end{aligned}$$

We have used induction, the inequality $\frac{1}{n-\delta} \leq \frac{1}{n-2\delta+k}$ (since $k \leq \delta$), and monotonicity of $C_i(\cdot)$: $C_i(\ell) \leq C_i(\ell+1)$. \square

Combinatorial Interpretation of $C_k(m)$. It turns out that $C_k(m)$ has a combinatorial interpretation, leading to the closed form in Lemma 8. For $n \in \mathbf{N}$, $k \in \mathbf{N}_0$, we consider the set of permutations \mathcal{S}_n , and we define the set $\mathcal{S}_n^{(k)}$ of k -decorated permutations. A k -decorated permutation is a pair $(\pi, (j_1, j_2, \dots, j_{h-1}))$, where $\pi \in \mathcal{S}_n$, h is the number of cycles of π , and j_1, j_2, \dots, j_{h-1} are positive integers summing up to k .

Assume some canonical ordering¹⁶ of the cycles of a permutation in \mathcal{S}_n , where the last cycle is the one containing n . Then we can view a k -decorated permutation in $\mathcal{S}_n^{(k)}$ as a permutation whose i th cycle is labeled by a positive integer j_i , except for the last one, which gets 0, and the sum of all the labels is k . Here comes the combinatorial interpretation.

Lemma 9. $k, m \in \mathbf{N}_0$.

$$m! C_k(m) = \#\mathcal{S}_{m+1}^{(k)} .$$

Proof. We argue that $c_k(m) := \#\mathcal{S}_{m+1}^{(k)}$ satisfies the same recurrence relation as $m! C_k(m)$.

$$c_0(m) = m! = m! C_0(m) , \quad \text{for } m \in \mathbf{N}_0 ,$$

because we can 0-decorate a permutation in \mathcal{S}_{m+1} only if there is exactly one cycle; there are $m!$ such permutations in \mathcal{S}_{m+1} .

$$c_k(0) = 0 = 0! C_k(0) , \quad \text{for } k \in \mathbf{N} ,$$

because the unique permutation in \mathcal{S}_1 has one cycle, and therefore cannot be k -decorated with positive k .

¹⁶ For example, order the cycles in increasing order according to their largest element.

For $m, k \in \mathbf{N}$, we split $\mathcal{S}_{m+1}^{(k)}$ into two sets. The first set contains the k -decorated permutations with $m+1$ appearing in a cycle of its own. For each i -decorated permutation in $\mathcal{S}_m^{(i)}$, $i = 0, 1, \dots, k-1$, there is exactly one corresponding k -decorated permutation in \mathcal{S}_{m+1} with $m+1$ appearing in a cycle (labeled 0) of its own, and the cycle containing m labeled $k-i$. Consequently, the first set contains

$$\sum_{i=0}^{k-1} c_i(m-1) \quad (4)$$

k -decorated permutations.

The second set contains the k -decorated permutations where $m+1$ appears in some nontrivial cycle. There are m ways to integrate $m+1$ into existing cycles of some permutation in $\mathcal{S}_m^{(k)}$. We let the cycles containing m and $m+1$, resp. switch their labels (unless m and $m+1$ appear in the same cycle). Hence, this second set has size

$$m \cdot c_k(m-1). \quad (5)$$

Combining (4) and (5) we obtain

$$c_k(m) = m \cdot c_k(m-1) + \sum_{i=0}^{k-1} c_i(m-1),$$

the recurrence relation for $m! C_k(m)$. □

$i \in \mathbf{N}_0$. A permutation with $i+1$ cycles gives rise to exactly $\binom{k-1}{i-1}$ k -decorated permutations, since k can be written in that number of ways as an ordered sum of i positive integers¹⁷. Thus,

$$C_k(m) = \frac{1}{m!} \sum_{i=0}^m \binom{k-1}{i-1} \left[\begin{matrix} m+1 \\ i+1 \end{matrix} \right], \quad \text{for } k, m \in \mathbf{N}_0,$$

and the identity in Lemma 8 is proven.

Upper Bound for $C_k(m)$. For the upper estimate we start using Lemma 10 in the appendix.

$$\begin{aligned} C_k(m) &= \frac{1}{m!} \sum_{i=0}^k \binom{k-1}{i-1} \left[\begin{matrix} m+1 \\ i+1 \end{matrix} \right] \\ &\leq \frac{1}{m!} \sum_{i=0}^k \binom{k-1}{i-1} \frac{m!}{i!} (H_m)^i \\ &= \sum_{i=0}^k \binom{k-1}{i-1} \frac{(H_m)^i}{i!} \leq \sum_{i=0}^k \binom{k}{i} \frac{(H_m)^i}{i!} \end{aligned}$$

¹⁷ Note that, indeed, 0 can be written in $1 = \binom{-1}{-1}$ ways as the sum of 0 positive integers.

$$\begin{aligned}
&\leq \sum_{i=0}^k \frac{(kH_m)^i}{(i!)^2} \quad \text{since } \binom{k}{i} \leq k^i/i! \\
&= \sum_{i=0}^k \left(\frac{(\sqrt{kH_m})^i}{i!} \right)^2 \leq \left(\sum_{i=0}^{\infty} \frac{(\sqrt{kH_m})^i}{i!} \right)^2 \\
&= e^{2\sqrt{kH_m}}.
\end{aligned}$$

This establishes the upper bound claimed in Lemma 8.

Theorem 4. $n \in \mathbf{N}$; $P \subseteq \mathbf{R}^d$, $\#P = n \geq 2(d+1)$.

A call $\text{Implicit}(P, \emptyset)$ computes a basis of P with an expected number of at most

$$e^{2\sqrt{(d+1)H_{n-d-1}}} \quad \text{basis}_{2(d+1)}^d\text{-computations (pivot steps, resp.)}$$

and an expected number of at most

$$(n - 2(d+1)) \min\{e^{2\sqrt{(d+1)H_{n-d-1}}}, 2^{d+1}\} \quad \text{inclusion tests.}$$

The bound for inclusion tests follows directly from the fact that no basis is involved in more than $n - 2(d+1)$ inclusion tests, and the simple bound concluded right after setting up the recursion for $t_k(n)$.

5 Bibliographical Remarks and Discussion

We have already indicated (and the reader might have suspected) that the methods described in this tutorial apply to problems beyond smallest enclosing balls. In fact, there is a host of other problems for which we have a ‘Lemma 1’, the most prominent one being *linear programming*.

In the general framework of *LP-type problems* (which is actually motivated by the linear programming case), we rank subsets S of some ground set P by an objective function¹⁸ $w : 2^P \rightarrow \mathcal{O}$, \mathcal{O} an ordered set, and we require (P, w) to satisfy the conditions of Lemma 1 (2)–(4).

Definition 7 (LP-type problem). P finite set; $w : 2^P \mapsto \mathcal{O}$, \mathcal{O} ordered set; $\delta \in \mathbf{N}_0$.

(P, w) is an LP-type problem of combinatorial dimension at most δ if the following three axioms are satisfied for all $S \subseteq P$.

- (1) If $S' \subseteq S$ then $wS' \leq wS$.
- (2) Let $S' \subseteq S$. Then $wS' = wS$ iff $w(S' \cup \{p\}) = wS'$ for all $p \in S \setminus S'$.
- (3) If $wS' \neq wS$ for all proper subsets S' of S , then $\#S \leq \delta$.

There are other abstractions of linear programming along these lines. Kalai has introduced so-called *abstract objective functions*, which are special LP-type problems [12]. There is also the framework of *abstract optimization problems* which is still more general than that of LP-type problems [8].

¹⁸ $wS = \rho S$ in case of smallest enclosing balls.

By Lemma 1, the smallest enclosing ball problem gives rise to an LP-type problem. In linear programming, P is a set of inequality constraints (halfspaces in \mathbf{R}^d), and for $S \subseteq P$, wS is defined as the lexicographically smallest point in the intersection of all halfspaces in S , where w -values are to be compared lexicographically [14].

Another interesting (nonlinear) LP-type problem is obtained from a set P of points in \mathbf{R}^d , by letting $wS := -\inf_{x \in \text{conv}S} \|x\|$ (i.e. the negative distance between the origin and $\text{conv}S$), [14,8]. Many more geometric optimization problems fit into the framework [9].

The significance of the abstract framework stems from the fact that the algorithm **Implicit** we have described above for smallest enclosing balls actually works (with the same analysis) for arbitrary LP-type problems, provided the problem-specific primitives are supplied:

- *Violation test* ‘ $w(C \cup \{p\}) > w(C)$ ’.
- *Pivot step* ‘ $\text{pivot}(B, p)$ ’: Given a basis B and an element p with $w(B \cup \{p\}) > wB$, it returns a basis $B' \subseteq B \cup \{p\}$ with $wB' > wB$.

In addition, the call to $\text{basis}_{2\delta}^d S$ needs to be replaced by some method for finding a basis of a set with at most 2δ elements. Even if this is done by brute force, **Implicit** is an expected $O(n)$ algorithm for solving an LP-type problem of constant combinatorial dimension over an n -element ground set. This is remarkable, because even for concrete LP-type problems like linear programming, linear-time algorithms have only been found in the eighties. Most notably, Megiddo published the first (deterministic) $O(n)$ algorithm for linear programming in fixed dimension in 1984 [15], with a 2^d dependence on the dimension d . A generalization of the $O(n)$ bound to certain convex programming problems, including the smallest enclosing ball problem, is due to Dyer [4].

By using randomization, simple $O(n)$ algorithms for linear programming have then been found by Clarkson [2] and Seidel [17]. Subsequent developments mainly consisted of attempts to reduce the dependence of the runtime on the dimension d . For linear programming, the first subexponential bounds have independently been found by Kalai [13] as well as Matoušek, Sharir and Welzl [14]. The notion of k -decorated permutations we have used in the analysis of the subexponential algorithm **Implicit** is also due to Kalai [12].

Combining these algorithms with the ones by Clarkson, one obtains the fastest known combinatorial algorithm for linear programming, of runtime (i.e. number of arithmetic operations)

$$O(d^2 n) + e^{O(\sqrt{d \log d})} ,$$

see [9]. The currently best deterministic bound is $O(d^{O(d)} n)$ [1].

As far as other LP-type problems (in particular the smallest enclosing ball problem) are concerned, the situation is similar, although there are additional technical obstacles.

Exactly these obstacles made us formulate the procedure **Implicit**(S, B) in such a way that the recursion bottoms out when $\#S \leq 2\delta$. The reader might have noticed that as long as there are elements in $S \setminus B$, we could continue

with the recursive call $\text{Implicit}(S \setminus \{p\}, B)$, for p randomly chosen in $S \setminus B$. In fact, we can, but then a subexponential bound for the expected number of basis computations does no longer hold. For linear programming, it still works out: because *every* basis has size d in the above formulation of the problem, the recursion bottoms out when $\#S = d$, in which case the basis computation is trivial; this ‘saves’ the subexponential bound, and the algorithm described in [14] is actually this variant of the algorithm **Implicit**.

In the general case, a problem of size δ is not trivial. On the contrary, in case of smallest enclosing balls, we have made quite some effort to solve a problem involving $\delta = d + 1$ points with an expected number of $O((\frac{3}{2})^d)$ steps.

The subexponential bound can be saved in general, though, because there is an algorithm to solve an LP-type problem over $O(\delta)$ elements with an expected number of $e^{O(\sqrt{\delta})}$ primitive operations by Gärtner [8]. This algorithm is more complicated than the ones discussed here, but it lets us obtain essentially the same bounds for arbitrary LP-type problems as we have them for linear programming.

The procedures **Explicit** as well as **ForceOrNot** are interesting in the sense that they do *not* work for general LP-type problems. **Explicit** has first been described in [19], and it is actually modeled after Seidel’s linear programming algorithm [17]. In order for these algorithms to work, we need a notion of explicitly ‘forcing’ elements on the ‘boundary’. Such a notion does not exist for general LP-type problems. However, the framework can be specialized to include problems for which this forcing makes sense, in which case the algorithms (that can then be interpreted as primal-dual methods) apply [10].

A An Estimate for Cycle Numbers

Lemma 10. $i, n \in \mathbb{N}_0; i \leq n$.

$$\begin{bmatrix} n+1 \\ i+1 \end{bmatrix} \leq \frac{n!}{i!} (H_n)^i .$$

Proof by induction.

$i = 0$:

$$\begin{bmatrix} n+1 \\ 1 \end{bmatrix} = n! = \frac{n!}{0!} (H_n)^0 .$$

$i = n$:

$$\begin{bmatrix} n+1 \\ n+1 \end{bmatrix} = 1 \leq \frac{n!}{n!} (H_n)^n .$$

$0 < i < n$:

$$\begin{aligned} \begin{bmatrix} n+1 \\ i+1 \end{bmatrix} &= n \begin{bmatrix} n \\ i+1 \end{bmatrix} + \begin{bmatrix} n \\ i \end{bmatrix} \\ &\leq n \frac{(n-1)!}{i!} (H_{n-1})^i + \frac{(n-1)!}{(i-1)!} (H_{n-1})^{i-1} \end{aligned}$$

$$\begin{aligned}
&= \frac{n!}{i!} \left((H_{n-1})^i + i \frac{1}{n} (H_{n-1})^{i-1} \right) \\
&\leq \frac{n!}{i!} \left(H_{n-1} + \frac{1}{n} \right)^i = \frac{n!}{i!} (H_n)^i,
\end{aligned}$$

where we employed the Binomial Theorem. \square

References

1. Bernard Chazelle, Jiří Matoušek: On linear-time deterministic algorithms for optimization problems in fixed dimensions, *J. Algorithms* **21** (1996), 579–597.
2. Kenneth L. Clarkson: A Las Vegas algorithm for linear and integer programming when the dimension is small, *J. Assoc. Comput. Mach.* **42**(2) (1995), 488–499.
3. Martin E. Dyer: Linear algorithms for two and three-variable linear programs, *SIAM J. Comput.* **13** (1984), 31–45.
4. Martin E. Dyer: On a multidimensional search technique and its application to the Euclidean one-center problem, *SIAM J. Comput.* **15** (1986), 725–738.
5. Jürgen Eckhoff: Helly, Radon, and Carathéodory type theorems, in *Handbook of Convex Geometry* (P.M.Gruber, J.M.Wills, eds.), Vol. **A** (1993), 389–448, North Holland.
6. Bernd Gärtner, Michael Hoffmann, Sven Schönherr: Geometric optimization, *Reference Manual of the Computational Geometry Algorithms Library (CGAL)*, Release 2.2 (www.cgal.org) (2000).
7. Bernd Gärtner, Sven Schönherr: An efficient, exact, and generic quadratic programming solver for geometric optimization, *Proc. 16th Ann. ACM Symp. Computational Geometry* (2000), 110–118.
8. Bernd Gärtner: A subexponential algorithm for abstract optimization problems, *SIAM J. Comput.* **24** (1995), 1018–1035.
9. Bernd Gärtner, Emo Welzl: Linear Programming – Randomization and abstract frameworks, *Proc. 13th Ann. ACM Symp. Theoretical Aspects of Computer Science* (1996), 669–687.
10. Bernd Gärtner, Emo Welzl: LP-type problems of the second kind for primal-dual methods, manuscript, in preparation (2001).
11. Ronald L. Graham, Donald E. Knuth, Oren Patashnik: *Concrete Mathematics; A Foundation for Computer Science*, Addison-Wesley (1989).
12. Gil Kalai: Linear programming, the simplex algorithm and simple polytopes, *Math. Programming* **79** (1997), 217–233.
13. Gil Kalai: A subexponential randomized simplex algorithm, *Proc. 24th Ann. ACM Symp. Theory of Computing* (1992), 475–482.
14. Jiří Matoušek, Micha Sharir, Emo Welzl: A subexponential bound for linear programming, *Algorithmica* **16** (1996), 498–516.
15. Nimrod Megiddo: Linear programming in linear time when the dimension is fixed, *J. Assoc. Comput. Mach.* **31** (1984), 114–127.
16. Rajeev Motwani, Prabhakar Raghavan: *Randomized Algorithms*, Cambridge University Press (1995).
17. Raimund Seidel: Small-dimensional linear programming and convex hulls made easy, *Discrete Comput. Geom.* **6** (1991), 423–434.
18. James Joseph Sylvester: A question in the geometry of situation, *Quart. J. Math.* **1** (1857), 79.
19. Emo Welzl: Smallest enclosing disks (balls and ellipsoids), in “New Results and New Trends in Computer Science”, (H. Maurer, ed.), *Lecture Notes in Computer Science* **555** (1991), 359–370.

Codes over Z_4

Tor Helleseeth

Department of Informatics, University of Bergen
Høyteknologisenteret, N-5020 Bergen, Norway
torh@ii.uib.no

Abstract. One recent direction in coding theory has been to study linear codes over the alphabet Z_4 and apply the Gray map from Z_4 to binary pairs to obtain binary nonlinear codes better than comparable binary linear codes. This connection between linear codes over Z_4 and nonlinear binary codes was also the breakthrough in solving an old puzzle of the apparent duality between the nonlinear Kerdock and Preparata codes. We present a description of this puzzle and a brief introduction to codes over Z_4 .

1 Introduction

In an important paper, Hammons et. al. [3] show how to construct well known binary nonlinear codes like Kerdock codes and Delsarte-Goethals codes by applying the Gray map to linear codes over Z_4 . Further, they solve an old open problem in coding theory that the weight enumerators of the nonlinear Kerdock codes and Preparata codes satisfy the MacWilliams identities. These discoveries as well as related work by Nechaev [6] lead to a strong interest in Z_4 -linear codes, and recently several other binary nonlinear codes which are better than comparable binary linear codes have been found using the Gray map on Z_4 -linear codes.

Many of the new codes are constructed from extended cyclic codes over Z_4 . These codes can be naturally described using the structure of a Galois ring instead of a Galois field which is normally used in describing binary linear codes. The purpose of this paper is to survey some of the recent developments in this area and describe the Z_4 -linear Kerdock and Preparata codes as well as some other related codes with excellent distance properties.

In Section 2 we give a brief background in coding theory. In Section 3 we give some preliminaries on Galois rings. In Section 4 we briefly describe the connections between Kerdock and Preparata codes found by Hammons et. al. [3] and mention some examples of some other good nonlinear codes. Techniques from Galois rings are applied to illustrate the proof of the minimum distance of the Preparata code over Z_4 .

2 Coding Theory

A binary (n, M, d) code C is a subset of F_2^n (the set of binary n -tuples) of size M and minimum Hamming distance d , where the Hamming distance between two vectors is the number of components for which they differ.

The code C is said to be *linear* if C is a subspace of dimension k . The dual code C^\perp of the $(n, 2^k, d)$ linear code C is the linear $(n, 2^{n-k}, d^\perp)$ code defined by

$$C^\perp = \{\mathbf{x} \in F_2^n \mid \mathbf{x} \cdot \mathbf{y} = 0 \text{ for all } \mathbf{y} \in C\},$$

where $\mathbf{x} \cdot \mathbf{y} = \sum_{i=0}^{n-1} x_i y_i \pmod{2}$ is the inner product between \mathbf{x} and \mathbf{y} . A linear code can be realized as the rowspace of a *generator matrix* G .

For practical applications codes with a large minimum distance are desirable. It is therefore important to find the minimum distance of the code. To find the minimum distance of a nonlinear code one has to compute the distance between all pair of codewords. This is more complicated than for linear codes where it suffices to find the minimum weight among all the nonzero codewords.

A code is said to be *distance invariant* if the set of distances from a codeword to all the other codewords is independent of the codeword. Clearly, a linear code is distance invariant. In this case the set of distances from a codeword is the set of weights of the codewords (i.e., the distances from the zero codeword).

The weight distribution of a code is therefore of fundamental interest. Let $A(z) = \sum_{i=0}^n A_i z^i$ and $B(z) = \sum_{i=0}^n B_i z^i$, where A_i and B_i denote the number of codewords of Hamming weight i in C and C^\perp respectively. For binary *linear* codes the MacWilliams identities hold, i.e.,

$$B(z) = \frac{1}{|C|} (1+z)^n A\left(\frac{1-z}{1+z}\right).$$

The lack of structure in a nonlinear code makes it more difficult to decode in an efficient manner than a linear code. There are, however, some advantages to nonlinear codes. For given values of the length n and minimum distance d it is sometimes possible to construct nonlinear codes with more codewords than is possible for linear codes. For example, for $n = 16$ and $d = 6$ the best linear code has dimension $k = 7$, i.e., it contains 128 codewords.

In 1967, Nordstrom and Robinson found a nonlinear code with parameters $n = 16$ and $d = 6$ containing $M = 256$ codewords, which has twice as many codewords as the best linear code for the same values of n and d .

In 1968, Preparata [7] generalized this construction to an infinite family of codes having parameters

$$(2^{m+1}, 2^{2^{m+1}-2m-2}, 6), \quad m \text{ odd}, m \geq 3.$$

A few years later, in 1972, Kerdock [5] gave another generalization of the Nordstrom-Robinson code and constructed another infinite class of codes with parameters

$$(2^{m+1}, 2^{2m+2}, 2^m - 2^{\frac{m-1}{2}}), \quad m \text{ odd}, m \geq 3.$$

The Preparata code contains twice as many codewords as the best known linear code and is optimal in the sense of having the largest possible size for the given length and minimum distance. The Kerdock code appears to have twice as many codewords as the best known linear code. In the case $m = 3$ the Preparata code and the Kerdock codes both coincide with the Nordstrom-Robinson code.

The Preparata and Kerdock codes are *distance invariant*. In particular, since they contain the all-zero codeword, their weight distribution equals their distance distribution.

In general there is no natural way to define the dual code of a nonlinear code, and thus the MacWilliams identities have no meaning for nonlinear codes. However, one can define the weight enumerator polynomial $A(z)$ of a nonlinear code in the same way as for linear codes. Even if the concept of a dual of a nonlinear code has no meaning, one can always take the weight enumerator $A(z)$ of a nonlinear code and compute its *formal dual* $B(z)$ from the MacWilliams identities.

An observation that puzzled the coding theory community for a long time was that the weight enumerator of the Preparata code $A(z)$ and the weight enumerator of the Kerdock code $B(z)$ satisfied the MacWilliams identities and in this sense these *nonlinear* codes “behaved like dual linear codes”.

In 1994, Hammons, Kumar, Calderbank, Sloane and Sole [3] gave an explanation of this puzzle. They gave a significantly simpler description of the family of Kerdock codes, and also defined a family of “Preparata” codes with the same weight distribution as the Preparata codes. In order to describe these and other related codes, we give a short introduction to the theory of Galois rings.

3 Galois Rings

Let Z_l denote the ring of integers modulo l . Let $\mu : Z_4 \rightarrow Z_2$ denote the modulo 2 reduction map on Z_4 , i.e.,

$$\mu(l) = l \pmod{2}.$$

If $h(x) = \sum_{i=0}^d h_i x^i$ is a polynomial over Z_4 , we define $\mu(h)$ to be the polynomial over F_2 given by $\mu(h) = \sum_{i=0}^d \mu(h_i) x^i$. For example,

$$\mu(x^3 + 2x^2 + x + 3) = x^3 + x + 1.$$

If $f \in Z_4[x]$ is monic and $\mu(f)$ is irreducible, we say that f is a *basic irreducible*. Thus the polynomial $x^3 + 2x^2 + x + 3$ is an example of a basic irreducible polynomial.

Let \bar{f} be a primitive polynomial over F_2 of degree m (i.e., an irreducible polynomial such that its zeros have order $2^m - 1$ and thus generate the nonzero elements in the finite field F_{2^m}). Let the polynomial over Z_4 be defined by

$$f(x^2) = (-1)^m \overline{f(x)} \overline{f(-x)}$$

where on the right hand side we regard \overline{f} , as a polynomial over Z_4 with $\{0, 1\}$ coefficients. Then since $\mu(f) = \overline{f}$, it follows that $f(x)$ is a basic irreducible over Z_4 .

Example. Let $\overline{f(x)} = x^3 + x + 1$, then this gives

$$\begin{aligned} f(x^2) &= (-1)^3 \overline{f(x)} \overline{f(-x)} \\ &= -(x^3 + x + 1)(-x^3 - x + 1) \\ &= x^6 + 2x^4 + x^2 + 3 \end{aligned}$$

and therefore

$$f(x) = x^3 + 2x^2 + x + 3.$$

and

$$\mu(f) = x^3 + x + 1 = \overline{f(x)}.$$

Next, let $f(x)$ be a basic irreducible, derived from a primitive binary polynomial as above. Set $R_{4^m} = Z_4[x]/(f(x))$, then $R_{4^m} \cong Z_4[\beta]$, where β belongs to some extension ring of Z_4 and satisfies

$$f(\beta) = 0.$$

Clearly R_{4^m} contains 4^m elements and every element z in R_{4^m} can be expressed in the form

$$z = \sum_{i=0}^{m-1} z_i \beta^i, \quad z_i \in Z_4.$$

It is also clear that R_{4^m} is a commutative ring with identity under the usual definitions of addition and multiplication.

Let $\alpha = \mu(\beta)$, i.e., $\alpha = \beta \pmod{2}$. Then since $\overline{f(\alpha)} = 0$, α is a primitive element of F_{2^m} . Since α has order $n = 2^m - 1$, it follows that the order of β in R_{4^m} is at least n . Also, $\alpha^n = 1$ implies that $\beta^n = 1 + 2\theta$, where $\theta \in R_{4^m}$. Thus β has order either n or $2n$. Since,

$$f(\beta^2) = (-1)^n \overline{f(\beta)} \overline{f(-\beta)}$$

and

$$\overline{f(\beta)} = \overline{f(-\beta)} = 0 \pmod{2}$$

it follows that

$$\overline{f(\pm\beta)} = 2\delta, \delta \in R_{4^m}.$$

Thus $f(\beta^2) = 0$, and it follows that $\beta, \beta^2, \dots, \beta^{2^m-1}$ are also zeros of f . Since β^2 has order n and is also a root of f we can assume (by replacing β by β^2 if necessary), without loss of generality that β has order n .

Set

$$\mathcal{T} = \{0\} \cup \{1, \beta, \beta^2, \dots, \beta^{n-1}\}.$$

Note that since $\mu(\beta^i) = \alpha^i$, an element in \mathcal{T} is uniquely defined by its reduction modulo 2. Every element $x + 2y, x, y \in \mathcal{T}$ belongs to R_{4^m} . Now

$$x_1 + 2y_1 = x_2 + 2y_2 \text{ iff } x_1 = x_2 \text{ and } y_1 = y_2.$$

This can be seen by first reducing modulo 2 which forces $x_1 = x_2$. Subtracting their common value from both sides gives $y_1 = y_2$. Thus every element z in R_{4^m} has a unique expression of the form

$$z = x + 2y, x, y \in \mathcal{T}.$$

The set \mathcal{T} is commonly referred to as the set of Teichmuller representatives or simply as the Teichmuller set.

If

$$(x_1 + 2y_1) + (x_2 + 2y_2) = x + 2y, \text{ where } x, y, x_i, y_i \in \mathcal{T},$$

then by raising both sides to the 2^m power, we see that

$$x = x_1 + x_2 + 2\sqrt{x_1 x_2} \text{ and } y = y_1 + y_2 + \sqrt{x_1 x_2}.$$

This is how addition is performed in R_{4^m} .

4 Kerdock and Preparata Codes over Z_4

In 1994, Hammons, Kumar, Calderbank, Sloane and Sole [3] gave a significantly simpler description of the family of Kerdock codes. They constructed a linear code over $Z_4 = \{0, 1, 2, 3\}$, which is an analogue of the binary 1-st order Reed-Muller code. This code is combined with a mapping called the Gray map that maps the elements in Z_4 into binary pairs. The Gray map ϕ is defined by

$$\phi(0) = 00, \phi(1) = 01, \phi(2) = 11, \phi(3) = 10.$$

The Lee weight of an element in Z_4 is defined by

$$w_L(0) = 0, w_L(1) = 1, w_L(2) = 2, w_L(3) = 1.$$

Extending ϕ in a natural way to a map $\phi : Z_4^n \rightarrow Z_2^{2n}$ one observes that ϕ is a distance preserving map from Z_4^n (under the Lee metric) to Z_2^{2n} (under the Hamming metric).

A linear code over Z_4 is a subset of Z_4^n such that any linear combination of two codewords is again a codeword. From a linear code \mathcal{C} of length n over Z_4 , one obtains a binary nonlinear code $C = \phi(\mathcal{C})$ of length $2n$ by replacing each component in a codeword in \mathcal{C} by its image under the Gray map.

The Lee weight distribution of the code \mathcal{C} over Z_4 equals the Hamming weight distribution of its binary image under the Gray map. In particular, the minimum Hamming distance of C equals the minimum Lee distance of \mathcal{C} and is equal to the minimum Lee weight of \mathcal{C} since \mathcal{C} is linear over Z_4 .

To obtain the Nordstrom-Robinson code we will construct a code over Z_4 of length 8 and then apply the Gray map. The starting point is the binary primitive polynomial $\overline{f(x)} = x^3 + x + 1$ of degree $m = 3$. Next, one forms the polynomial $f(x)$ with coefficients in Z_4 given by

$$\begin{aligned} f(x^2) &= (-1)^m \overline{f(x)} \overline{f(-x)} \pmod{4} \\ &= x^6 + 2x^4 + x^2 + 3 \pmod{4}. \end{aligned}$$

Let β be a zero of $f(x) = x^3 + 2x^2 + x + 3 \in Z_4[x]$, that is $\beta^3 + 2\beta^2 + \beta + 3 = 0$. Then we can express all powers of β in terms of $1, \beta$ and β^2 , as follows,

$$\begin{aligned}\beta^3 &= 2\beta^2 + 3\beta + 1 \\ \beta^4 &= 3\beta^2 + 3\beta + 2 \\ \beta^5 &= \beta^2 + 3\beta + 3 \\ \beta^6 &= \beta^2 + 2\beta + 1 \\ \beta^7 &= 1.\end{aligned}$$

Consider the code \mathcal{C} over Z_4 with generator matrix given by

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \beta & \beta^2 & \beta^3 & \beta^4 & \beta^5 & \beta^6 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 2 & 3 & 1 \\ 0 & 0 & 1 & 0 & 3 & 3 & 3 & 2 \\ 0 & 0 & 0 & 1 & 2 & 3 & 1 & 1 \end{bmatrix},$$

where the column corresponding to β^i is replaced by the coefficients in its expression in terms of $1, \beta$, and β^2 . Then the Nordstrom-Robinson code is the Gray map of \mathcal{C} . Note that \mathcal{C} is exactly the Kerdock code \mathcal{K}_m over Z_4 when $m = 3$, which will be described later.

The dual code \mathcal{C}^\perp of a code \mathcal{C} over Z_4 is defined similarly as for binary linear codes, except that the inner product of the vectors $\mathbf{x} = (x_1, x_2, \dots, x_n)$ and $\mathbf{y} = (y_1, y_2, \dots, y_n)$ with components in Z_4 is defined by

$$(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n x_i y_i \pmod{4}.$$

The dual code \mathcal{C}^\perp of \mathcal{C} is then

$$\mathcal{C}^\perp = \{\mathbf{x} \in Z_4^n \mid \mathbf{x} \cdot \mathbf{c} = 0 \text{ for all } \mathbf{c} \in \mathcal{C}\}$$

For a linear code \mathcal{C} over Z_4 there is a MacWilliams relation that determines the complete weight distribution of the dual code \mathcal{C}^\perp from the complete weight distribution of \mathcal{C} . Therefore, one can compute the relation between the Hamming weight distribution of the nonlinear codes $\mathcal{C} = \phi(\mathcal{C})$ and $\mathcal{C}_\perp = \phi(\mathcal{C}^\perp)$, and it turns out that the MacWilliams identities hold.

Hence, to find nonlinear binary codes related by the MacWilliams identities one can start with a pair of Z_4 -linear dual codes and apply the Gray map. For any odd integer $m \geq 3$, the Gray map of the code \mathcal{K}_m over Z_4 with generator matrix

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & \beta & \beta^2 & \dots & \beta^{2^m-2} \end{bmatrix}$$

is the binary nonlinear $(2^{m+1}, 2^{2m+2}, 2^m - 2^{\frac{m-1}{2}})$ Kerdock code. The Gray map of \mathcal{K}_m^\perp has the same weight distribution as the $(2^{m+1}, 2^{2^{m+1}-2m-2}, 6)$ Preparata code. It is, however, not identical to the Preparata code and is therefore denoted

the “Preparata” code. Hence the Kerdock code and the “Preparata” code are the Z_4 -analogy of the 1-st order Reed Muller code and the extended Hamming code respectively.

To illustrate the techniques using Galois rings, we will give a proof of the minimum distance of the Preparata code aswell as a subcode of the Preparata code known as the Goethals code. Let X, Y, A, B denote elements in \mathcal{T} and x, y, a, b the corresponding elements in F_{2^m} after using the modulo 2 projection μ . We state two useful lemmas. The first result appeared earlier in connection with our discussion on Galois rings and is repeated here for convenience.

Lemma 1. *Let $X, Y \in \mathcal{T}$ satisfy $X + Y = A + 2B$, where $A, B \in \mathcal{T}$. Then $A = X + Y + 2\sqrt{XY}$ and $B = \sqrt{XY}$ or equivalently $a = x + y$ and $b^2 = xy$.*

Lemma 2. *Let m be an odd integer. Then the zeros of the polynomials $p(x) = x^4 + ax + b$ in $F_{2^m}[x]$ cannot all be distinct and belong to F_{2^m} .*

Proof. Let us assume that $p(x) = x^4 + ax + b$ has four zeros in F_{2^m} . Since $p(x)$ is an affine polynomial, then $x^4 + ax$ has four zeros, i.e., $x^3 + a$ has three zeros in F_{2^m} , which is impossible since m odd implies that $\gcd(3, 2^m - 1) = 1$.

Theorem 1. *The minimum Lee distance of the quaternary Preparata \mathcal{P} code with parity check polynomial*

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & \beta & \beta^2 & \cdots & \beta^{2^m-2} \end{bmatrix}$$

is 6.

Proof. Let \mathbf{c} be a nonzero codeword in \mathcal{P} . By definition $\mu(\mathbf{c})$ is a codeword in the extended Hamming code with generator polynomial being the minimum polynomial of α , where $\alpha = \mu(\beta)$. Therefore, if $\mu(\mathbf{c}) = \mathbf{0}$ then \mathbf{c} has Hamming weight ≥ 4 and thus Lee weight ≥ 8 . If $\mu(\mathbf{c}) \neq \mathbf{0}$ there are only two possibilities, for a codeword \mathbf{c} of weight < 6 which we will exclude.

Case I (\mathbf{c} is of type $0^{2^m-4}1^2(-1)^2$). Let X_1, X_2 denote the locations with error values 1 and let X_3, X_4 denote the locations with error values -1 . This leads to the syndrome equation $X_1 + X_2 = X_3 + X_4$. From Lemma 1 we get $x_1 + x_2 = x_3 + x_4$ and $x_1x_2 = x_3x_4$. Hence the quadratic polynomial $f(x) = (x - x_1)(x - x_2) = (x - x_3)(x - x_4)$ has four distinct zeros in F_{2^m} , a contradiction.

Case II (\mathbf{c} is of type $0^{2^m-4}1^4$ (the case $0^{2^m-4}(-1)^4$ is similar)). In this case we get a syndrome equation $X_1 + X_2 + X_3 + X_4 = 0$. Using Lemma 1 this leads to the two equations $x_1 + x_2 + x_3 + x_4 = 0$ and $x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4 = 0$. Let $\sigma(x) = (x - x_1)(x - x_2)(x - x_3)(x - x_4)$, then $\sigma(x) = x^4 + ax + b$ for some $a, b \in F_{2^m}$. Lemma 2 implies that $\sigma(x)$ cannot have four distinct zeros in F_{2^m} , a contradiction.

With this, we have shown that the minimum Lee weight of the Preparata code is at least 6. Consider next, the Z_4 2^m -tuple \mathbf{a} having 4 ones in locations corresponding to $\{0, X, Y, Z\}$ and zeros elsewhere, where $\{0, X, Y, Z\}$ are distinct

elements in \mathcal{T} that satisfy $x + y + z = 0$. Let $X + Y + Z = 2\theta, \theta \in \mathcal{T}$ and let \mathbf{b} be a second 2^m -tuple over Z_4 with 2's in locations $\{0, \theta\}$ and zero everywhere else. Then $\mathbf{a} + \mathbf{b}$ has Lee weight ≤ 6 and satisfies the parity check equations of the code \mathcal{P} .

Hammons, Kumar, Calderbank, Sloane and Sole [3] also showed that the binary code defined by $C = \phi(\mathcal{C})$, where \mathcal{C} is the quaternary code with parity-check matrix given by

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 0 & 1 & \beta & \beta^2 & \dots & \beta^{2^m-2} \\ 0 & 2 & 2\beta^3 & 2\beta^6 & \dots & 2\beta^{3(2^m-2)} \end{bmatrix}$$

is a binary nonlinear $(2^{m+1}, 2^{2^{m+1}-3m-2}, 8)$ code whenever $m \geq 3$ is odd. This code has the same weight distribution as the Goethals code which is a nonlinear code that has four times as many codewords as the comparable extended triple-error correcting primitive BCH code. The code $C_\perp = \phi(\mathcal{C}^\perp)$ is identical to a binary nonlinear code, that was constructed in a much more complicated way by Delsarte and Goethals more than twenty years ago.

A natural question has been to look for further similar constructions to find codes that are better than the best known nonlinear codes. Helleseth, Kumar and Shanbhag [4] showed that a Z_4 -linear code with the same weight distribution as the Goethals code is obtained if the rows $\{1, \beta, 2\beta^3\}$ of H are replaced by $\{1, \beta, 2\beta^{2^k+1}\}$, where $m \geq 3$ is odd and $\gcd(k, m) = 1$. The dual code has the same parameters as the Delsarte-Goethals code.

Calderbank and McGuire [1] observed that when $m = 5$, the Gray map of the code with parity-check matrix with rows $\{1, \beta, \beta^3, 2\beta^5\}$ is a $(64, 2^{37}, 12)$ code. The minimum distance of the best known binary linear code of the same length and size is only 10. Similarly, Calderbank et. al. [2] found that the nonlinear binary code obtained from the Z_4 -linear code with a parity-check matrix with rows $\{1, \beta, \beta^3, \beta^5\}$ is a $(64, 2^{32}, 14)$ code. The minimum distance of the best known binary linear code of the same length and size is only 12.

5 Conclusions

We have surveyed the puzzle of the MacWilliams duality of nonlinear Kerdock and Preparata codes and showed how the Kerdock codes can be obtained from linear codes over Z_4 using the Gray map. Some descriptions of some other very good codes obtained from Z_4 -linear codes are given.

References

1. A.R. Calderbank and G. McGuire, "Construction of a $(64, 2^{37}, 12)$ code via Galois rings", *Designs, Codes and Cryptography*, 10 (1997), 157-165.
2. A.R. Calderbank, G. McGuire, P.V. Kumar and T. Helleseth, "Cyclic codes over Z_4 , locator polynomials and Newton's identities", *IEEE Trans. Inform. Theory*, 42 (1996), 217-226.

3. A.R. Hammons Jr., P.V. Kumar, A.R. Calderbank, N.J.A. Sloane and P. Solé, "The Z_4 -linearity of Kerdock, Preparata, Goethals, and related codes", *IEEE Trans. Inform. Theory*, 40 (1994), 301-319.
4. T. Helleseth, P. V. Kumar and A. Shanbhag, "Exponential sums over Galois rings and their applications", *Finite Fields and their Applications, London Mathematical Society, Lecture Note Series 233*, edited by S. Cohen and H. Niederreiter, Cambridge University Press, (1996), 109-128.
5. A.M. Kerdock, "A class of low-rate nonlinear binary codes", *Inform. Contr.*, 20 (1972), 182-187.
6. A. Nechaev, "The Kerdock code in a cyclic form", *Discrete Math. Appl.*, 1 (1991), 365-384.
7. F.P. Preparata, "A class of optimum nonlinear double-error correcting codes", *Inform. Contr.*, 13 (1968), 378-400.

Degree Bounds for Long Paths and Cycles in k -Connected Graphs

Heinz Adolf Jung

Department of Mathematics, Technical University of Berlin
Straße des 17. Juni 136, 10623 Berlin
`jung@math.tu-berlin.de`

For vertices x, y of a connected graph G let $D_G(x, y)$ denote the length of a longest (x, y) -path in G . Strangely enough $D(\cdot, \cdot)$ defines a somewhat exotic metric on the vertex set $V(G)$. To see this, consider a longest (x, z) -path P and a longest (y, x) -path Q in G . For the first vertex v on Q in $V(P)$ we then have $D(y, z) \geq |Q(y, v)| + |P(v, z)|$ and $D(x, y) \geq |P(x, v)| + |Q(v, y)|$, consequently $D(x, y) + D(y, z) \geq 2|Q(y, v)| + D(x, z)$. Here $|P|$ and $|G|$ denote the number of vertices on P and in G respectively.

In 2-connected graphs G the parameter $D(G) = \min_{x \neq y} D_G(x, y)$ and the circumference $c(G) = \max(|C| : C \text{ cycle in } G)$ are closely related. Obviously $c(G) \geq D(G) + 1$, and it is easy to see that $D(G) \geq \frac{1}{2}c(G)$. Bondy and Locke showed that in 3-connected G moreover $c(G) \geq \frac{2}{5}L(G)$ where $L(G) = \max_{x, y} D_G(x, y)$ is the length of a longest path in G (see [1] and [6]).

In the first part of these notes we elaborate on algorithmic aspects related to the classical degree bounds for $c(G)$. While in general the determination of $c(G)$ is hard the standard proof for Theorem 1 below provides a good algorithm.

Theorem 1. *A graph G on $n \geq 3$ vertices with minimum degree $\delta \geq \frac{n}{2}$ has a hamiltonian cycle.*

This well-known basic result of G.A. Dirac [2] is the first degree bound for hamiltonicity and was the seed for an enormous offspring.

In the standard proof one starts with any path in the connected graph G and extends it by successively attaching edges to the end vertices of the path at hand. This procedure (*Ext1*) comes up with a path $Q = v_0 v_1 \dots v_q$ such that v_0 and v_q have only neighbors on Q . If $N(v_0)$ and $N^+(v_q) := \{v_{i+1} : v_i \in N(v_q)\}$ are disjoint, clearly $v_q \notin N(v_0)$ and $|Q| \geq 1 + d(v_0) + d(v_q)$. If $v_{i+1} \in N(v_0) \cap N^+(v_q)$ we obtain a cycle C with vertex set $V(C) = V(Q)$ and $P[v_0, v_i] \cup P[v_{i+1}, v_q] \subseteq C$. If C is not hamiltonian pick an edge $v_j w$ where $w \in V(G) - V(C)$ to obtain a path with vertex set $V(C) \cup \{w\}$ (*Ext2*). Using alternately *Ext1* and *Ext2* to their limit one eventually gets a path $P = P[x, y]$ such that

- (a) x, y are not adjacent and $|P| - 1 \geq d(x) + d(y)$ or
- (b) x, y are adjacent and $P \cup xy$ is a hamiltonian cycle of G . Observe that a graph on $n \geq k + 1$ vertices is k -connected if $d(x) + d(y) \geq n - k + 2$ for every pair of non-adjacent vertices x, y .

Good algorithms are also available to verify another (more general) basic result of Dirac.

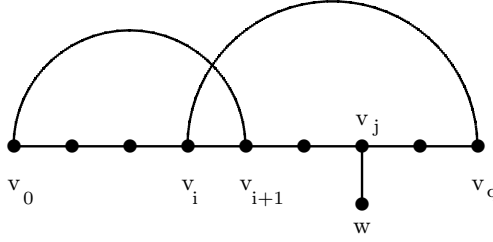


Fig. 1.

Theorem 2. *If G is 2-connected, then G has a hamiltonian cycle or $c(G) \geq 2\delta$.*

A proof can be based on a lemma by Erdős and Gallai [3] namely that $D(G) \geq \delta$ for 2-connected graphs. Also the “vine method” (see [1], [6]) can be used to devise a good algorithm towards Theorem 2 (cf. [7]). Here we sketch a good algorithm leading to the following (quite useful) slight strengthening of that lemma.

Lemma 1. *A 2-connected graph G contains vertices $a \neq b$ such that $D(G) \geq \max(d(a), d(b))$.*

Let us call a sequence of paths $P_1 = P_1[a_1, b_1], \dots, P_m[a_m, b_m]$ in G a *tower*, if $|P_i| \geq 3$, $P_i(a_i, b_i) = P_i[a_i^*, b_i^*] \subseteq G - (P_1 \cup \dots \cup P_{i-1})$ and $a_i, b_i \in P_{i-1} - (P_1 \cup \dots \cup P_{i-2})$ ($i = 2, \dots, m$). Rather $P_1 \cup \dots \cup P_m$ can be viewed as a building with *floors* $P_i[a_i^*, b_i^*]$. Let G_i denote the component of $G - (P_1 \cup \dots \cup P_{i-1})$ which contains $P_i[a_i^*, b_i^*]$. We say that the tower is *solid* if $N(G_i) \cap P_{i-1}(a_i, b_i) = \emptyset$ and a_i^* (or b_i^*) is inner vertex of G_i ($i = 2, \dots, m$). The latter means that a_i^* is contained in an end block B_i of G_i and not the unique cut vertex $c_i \in V(B_i)$ of G_i in case G_i has cut vertices. Mind that we allow for $|G_i| = 1$ which may only occur when $i = m > 1$. Observe also that the condition $N(G_i) \cap P_{i-1}(a_i, b_i) = \emptyset$ ($i = 2, \dots, m$) implies that $P_i(a_i, b_i)$ has no neighbors on $P_{i-1}(a_{i-1}, b_{i-1}) \cup \dots \cup P_2(a_2, b_2)$. We cannot go on building if $P_m[a_m^*, c_m]$ is a hamiltonian path of B_m and we then declare $P_m[a_m^*, b_m^*]$ the *roof* of the tower.

Suppose that the tower is solid but $|P_m[a_m^*, c_m]| < |B_m|$. Pick any component H_{m+1} of $B_m - P_m[a_m^*, c_m]$ and label $N(H_{m+1}) \cap P_m[a_m^*, c_m] = \{x_1, \dots, x_s\}$ in the order on P_m . We go on building with a path P_{m+1} of the form $P_{m+1}[x_j, x_{j+1}]$ where $P_{m+1}(x_j, x_{j+1}) = P_{m+1}[a_{m+1}^*, b_{m+1}^*]$ is a path in H_{m+1} . If $V(H_{m+1}) = \{w\}$ let $P_{m+1} = x_1 w x_2$. If $|H_{m+1}| \geq 2$ and H_{m+1} has no cut vertex we choose $x_j \neq x_s$ so that in addition $a_{m+1}^* \neq b_{m+1}^*$. If finally H_{m+1} has cut vertices we can choose x_j, x_{j+1} such that a_{m+1}^* or b_{m+1}^* (say a_{m+1}^*) is inner vertex of H_{m+1} , say in the block B of H_{m+1} while b_{m+1}^* is not inner vertex of H_{m+1} in $V(B)$.

Eventually we obtain a solid tower with roof. Say the above discussed tower P_1, \dots, P_m has roof $P_m[a_m^*, c_m]$. For our purposes we fancy a tower with decreasing floor sizes $|P_i[a_i^*, b_i^*]|$ towards the roof. If $|P_i| \geq |P_{i-1}[a_i, b_i]|$ we can replace $P_{i-1}[a_i, b_i]$ on P_{i-1} by P_i and then obtain another (a_{i-1}, b_{i-1}) -path P'_{i-1} . It turns out that $P_1, \dots, P_{i-2}, P'_{i-1}, P_{i+1}, \dots, P_m$ is again a solid tower with roof. We perform this destruction procedure from the roof downward as long as $m \geq 2$.

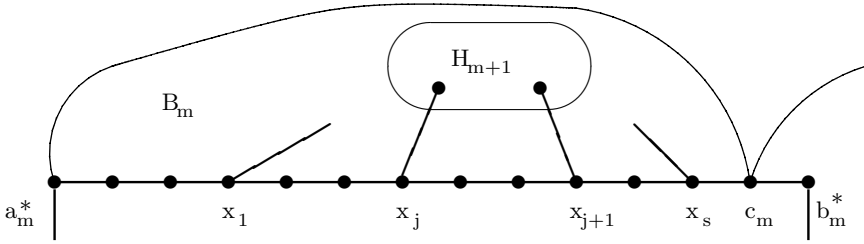


Fig. 2.

Now assume in addition $|P_i| < |P_{i-1}[a_i, b_i]|$ for all $i \geq 2$, if $m \geq 3$. If moreover $|P_2| < |P_1[a_2, b_2]|$, then $|P_1| \geq d(v) + 1 + 3(m-1)$ for all inner vertices v of G_m on $P_m[a_m^*, c_m]$. If $|P_2| \geq |P_1[a_2, b_2]|$ and hence $m = 2$, then another destruction produces an (a_1, b_1) -path P'_1 such that $|P'_1| \geq d(v) + 1$ for all inner vertices v of G_2 on $P_2[a_2^*, c_2]$.

Exploiting the ramification in the choice of P_m it is not difficult to construct non-adjacent a, b in Lemma 1 unless G is a complete graph.

In the second part we discuss the main obstructions standing against better degree bounds. Small connectivity is one of them as for example $G = K_2 + mK_{\delta-1}$ ($m \geq 2$) has minimum degree δ and $c(G) = 2\delta$. In general $G = K_k + mK_{\delta+1-k}$ ($m \geq k$) has connectivity $\kappa(G) = k$, minimum degree δ and $c(G) = k + k(\delta + 1 - k) = k\delta - k(k-2)$. Another problem present graphs G such that longest cycles C in G split $G - C$ into small components. For example $G = K_k + rK_1$ ($r \geq k$) has circumference $2k = 2\delta$ and longest cycles C in G are dominating (that is $E(G-C) = \emptyset$). More generally $G = K_k + rK_{s+1}$ ($r > k$) has connectivity $k = \delta - s$ and $c(G) = k(s+2) = k\delta - k(k-2)$ while longest cycles split off components isomorphic to K_{s+1} .

The next result shows how these obstructions can be handled - to a certain extent.

Theorem 3. *Let C be a longest cycle in the 2-connected graph G and let H be a component of $G - C$. Then H contains a vertex v with the following properties.*

- (1) $|C \cup H| \geq 3d(v) - 2$, if $|H| \geq 2$,
- (1') $|C| \geq 3d(v) - 3$, if $|H| \geq 2$ and G is 3-connected,
- (2) $|C \cup H| \geq 4d(v) - 5$, if $|H| \geq 3$ and G is 3-connected,
- (2') $|C| \geq 4d(v) - 8$, if $|H| \geq 3$ and G is 4-connected.

To indicate the approach we show (2') in the case when H is 2-connected. Fix a cyclic orientation in C and label $N_C(H) = \{x_1, \dots, x_s\}$ in order around C . As C is longest we have $|C(x_i, x_{i+1})| \geq 1$, moreover $|C(x_i, x_{i+1})| \geq D(H) + 1$, if x_i, x_{i+1} have distinct neighbors in H . Call $C[x_i, x_{i+1}]$ of the latter type a good segment. Letting q denote the number of good segments we obtain $|C| \geq 2s + qD(H)$. Determine vertices $a \neq b$ in H according to Lemma 1 (applied to H). If $q \geq 4$, then $2s + qD(H) = 2s + 2q + 4D(H) - 8 + (q-4)(D(H)-2)$. As $q \geq |N_C(a) \cap N_C(b)|$ indeed $|C| \geq 2d_C(a) + 2d_C(b) + 2d_H(a) + 2d_H(b) - 8$. If

$q = 3$ and G is 4-connected, necessarily $|H| = 3$ and $|N_C(a) \cap N_C(b)| \leq 3$. In this case clearly again $|C| \geq 2d_C(a) + 2d_C(b) - 6 + 3D(H) = 2d(a) + 2d(b) - 8$.

We refrain from treating the case when H has cut vertices as it is handled similarly and on the other hand affords the introduction of further technical notation. For further results in the vein of Theorem 3 see [5]. Very recently Vumar and Jung extended (2') to 3-connected graphs by classifying the exceptional graphs.

Better bounds in many special graph classes have been communicated, for example in bipartite graphs and regular graphs, also in line graphs and more generally in claw-free graphs. The latter by definition are graphs without induced $K_{1,3} = K_1 + 3K_1$ (see the monograph [8] by Voss).

In view of Theorem 3 and the above examples one is tempted to conjecture that $c(G) \geq k\delta - k(k-2)$ holds whenever G is k -connected and $|H| \geq k-1$. However, for example $G = K_k + mK_{1,r}$ ($m \geq k \geq 4, r \geq 2$) has connectivity k and $c(G) = 4k = 4\delta - 4$ while longest cycles in G split off components isomorphic to $K_{1,r}$. But $L := L(K_{1,r}) = 2$ and hence $c(G) = (L+2)\delta - 4$.

In the third and last part we deal with obstructions of this general kind. To overcome those the definition of "small" component H off C has to be adjusted. It turns out - as suggested by the examples - that $L(H)$ is an appropriate measure.

Already in 1980 Bondy conjectured that a longest cycle C in a k -connected graph on $n \leq (k+1)\delta - k(k-1)$ vertices satisfies $L(G-C) < k-1$. In order to include Theorem 1 and (1) in Theorem 3 set $L(\emptyset) < 0$ and $L(G) = 0$ when $V(G) \neq \emptyset$ and $E(G) = \emptyset$. To be precise Bondy conjectured the somewhat stronger version involving $\frac{1}{k+1}\sigma_{k+1}$ instead of δ , where σ_{k+1} is the minimum degree sum $d(u_1) + \dots + d(u_{k+1})$ taken over all independent sets $\{u_1, \dots, u_{k+1}\}$ in G .

A natural extension refers to Theorem 2 and (2'), (4') in Theorem 3, namely that a longest cycle C in a k -connected graph G always satisfies $|C| \geq k\delta - k(k-2)$ or $L(G-C) < k-2$. The latter conjecture was settled affirmatively only last year by Min Aung and Jung.

Theorem 4. *Let C be a longest cycle in the k -connected graph. If $L(G-C) \geq k-2 \geq 0$, there exist distinct vertices u_1, \dots, u_{k-1} in $G-C$ such that $|C| \geq 2d(u_1) + d(u_2) + \dots + d(u_{k-1}) - k(k-2)$.*

In our approach we pick a component H of $G-C$ such that $L(G-C) = L(H)$. We then fix a longest path $P = v_1 \dots v_{l+1}$ in H and embed P in a DFS-tree T with root v_{l+1} . This means that adjacent vertices v, w in H are always comparable in the partial order \leq_T . Here we set $v \leq_T w$, if w is on the unique path $T[v, v_{l+1}]$ in T from v to v_{l+1} . The vertices u_1, \dots, u_{k-1} are constructed in H with a certain preference for vertices of degree 1 in T . This can be done so that $d(u_i) = 1$ or $u_i \in P$. A vertex $u_i \in T-P$ comes along with a path $S_i = T[u_i, w_i] \cup w_i v_i$ such that $S_i \cap P = v_i$. The construction can be performed so that the paths S_i are pairwise disjoint. Moreover, if $u_i = v_i$, then $N_H(u_i)$ has only elements on P and on paths S_j with $j < i$.

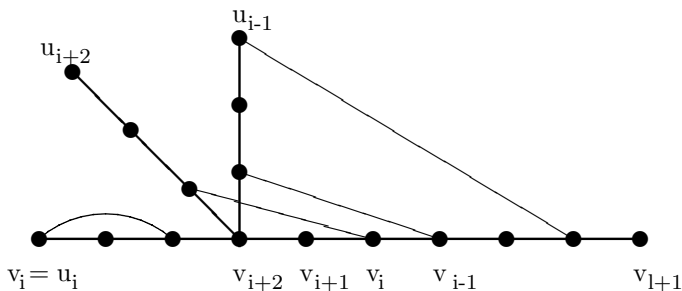


Fig. 3.

Once u_1, \dots, u_{k-1} are constructed $|C(x_i, x_{i+1})|$ is estimated in terms of $d_C(u_1), \dots, d_C(u_{k-1})$. Here again $N_C(H) = \{x_1, \dots, x_s\}$ as introduced in the context of Theorem 3.

Several cases have to be distinguished relating to the position of $N_H(x_i) \cup N_H(x_{i+1})$ in T . Unfortunately as yet the establishment of suitable bounds is in some cases quite involved and loaded with technical notions. This is beyond the scope of the present notes.

In future research also vertices outside H should be considered in the vein of the “classical” choice (x_1^+, \dots, x_s^+) and the choice in a fore-runner of Theorem 3 by Fraisse and Jung [4].

References

1. Bondy, J.A. and Locke, S.C.: Relative lengths of paths and cycles in 3-connected graphs. *Discrete Math.* 33 (1981), 111-122.
2. Dirac, G.A.: Some theorems on abstract graphs. *Proc. London Math. soc.* 3 (1952)2, 69-81.
3. Erdős, P. and Gallai, T.: On maximal paths and circuits in graphs. *Acta Math. Acad. Sci. Hung.* 10 (1959), 337-356.
4. Fraisse, P. and Jung, H. A.: Longest cycles and independent sets in k -connected graphs. In: *Recent Studies in Graph Theory* (edited by V.R. Kulli), Vishna Internat. Publ. Gulbarga, India, 1989, p. 114-139.
5. Jung, H.A.: Long cycles in graphs with moderate connectivity. In: *Topics in Combinatorics and Graph Theory, Essays in Honour of Gerhard Ringel* (edited by R. Bodendiek and R. Henn), Physica-Verlag, Heidelberg, 1990, p. 765-778.
6. Locke, S.C.: Relative lengths of paths and cycles in k -connected graphs. *J. Combin. Theory B* 32 (1982), 206-222.
7. Lovász, L.: *Combinatorial Problems and Exercises*. Akadémiai Kiadó, Budapest, 1979.
8. Voss, H.J.: *Cycles and Bridges in Graphs*, Kluwer Academic Publishers, Dordrecht, Netherlands.

Data Structures for Boolean Functions

BDDs – Foundations and Applications

Christoph Meinel and Christian Stangier

FB IV – Informatik, Universität Trier, 54286 Trier, Germany
`{meinel,stangier}@uni-trier.de`

Abstract. This article gives an outline of a lecture about ordered binary decision diagrams (OBDDs). Introduced in 1986, OBDDs nowadays are the state-of-the-art data structure for representation of Boolean functions in electronic design automation (EDA).

An overview is given about the data structure, its properties, algorithmic behaviour, and the most important applications. Proofs and technical details have been omitted but can be found all in [28].

1 Introduction

During the last years we are facing an enormous increase in system integration of VLSI (Very Large Scale Integration) designs. Moore's law, which says that the number of transistors on a chip doubles every 18 month is still valid and an end of this growth is not within view.

On the other hand there is a growing need for proving correctness of complex designs for the following two reasons:

1. Simulation is the conventional method for validation of digital designs. This method loses its ability to guarantee correctness if designs grow larger and become more complex.
2. Economic and security reasons request for mathematical (i.e. formal) proofs of the correctness of digital systems.

A central problem in CAD (computer aided design) of digital designs is the representation of Boolean functions. Within the last decade ordered binary decision diagrams have become one of the most popular data structures in this area.

The classical application for OBDDs is the formal verification of combinatorial circuits. Here, the equivalence of the specification and an implementation of a circuit has to be checked. Recently, the focus of the interest in research and industry has moved to the application of OBDDs to sequential systems. Many design tools use OBDDs to represent finite state machines. The emerging need for formal verification of sequential systems requires use of highly sophisticated data structures and algorithms. But, the application of OBDDs is not only limited to the representation of digital circuits of state sets. Other areas like representation of protocols or image compaction have been explored.

2 Data Structures for Boolean Functions

In computer-aided design, Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$ are of central importance for describing the switching behavior of digital circuits. Hence, those functions are also called *switching functions*. By introducing a suitable 0-1-encoding, all finite problems can – at least in principle – be modeled by means of switching functions. The great importance of switching functions stems from the possibility to obtain substantially simplified, optimized and with optional properties provided circuits by applying optimization techniques during the design process. In the area of VLSI circuits this task is performed by CAD systems. But, before optimization techniques can be applied, a way to represent the switching functions themselves uniquely and as efficiently as possible in computers has to be found.

The representation of a Boolean function is not an end in itself: On the basis of such a representation the various algorithms of CAD systems are performed. Hence, it is important to analyze the ability of the representation to support the basic operations of the algorithms.

The following list gives basic operations on Boolean functions ($f, g : \{0,1\}^n \rightarrow \{0,1\}$) in the field of CAD for VLSI design and verification usually called EDA:

Satisfiability test (SAT): Test whether there exists an assignment $a \in \{0,1\}^n$ of the variables of f with $f(a) = 1$.

Equivalence test: Check whether $f \equiv g$ for two functions f, g (alternatively (SAT($f \equiv g$))).

Evaluation: Compute $f(a_1, \dots, a_n)$ for a given assignment (a_1, \dots, a_n)

Synthesis: Compute a representation P_h for $h = f \otimes g$ for a given binary Boolean operator \otimes .

Replacement by function: Compute a representation P_h for $h = f|_{x_i=g}$ (i.e. the input x_i in f is replaced by the Boolean function g).

Another important operation on the representation of a Boolean function f is the **minimization** of the representation P_f of f .

Is it possible to work in EDA with any data structures for the representation of Boolean function or are special properties needed? For well known representations for Boolean functions it holds:

Circuits The elementary SAT problem is NP-complete for circuits and the minimization problem is NP-hard.

Boolean formulas The equivalence test is co-NP-complete for Boolean formulas.

Truth tables: All operations can be performed efficiently, but truth tables always require exponential space.

Conjunctive Normal Form (CNF): Even simple functions often have only an exponential representation in CNF. The SAT problem is NP-complete for CNF.

Disjunctive Normal Form (DNF): The problems with DNF are similar to CNF, but SAT is in P.

Branching Programs (BPs): Many functions can be represented very efficiently using branching programs but already the SAT problem is NP-complete for BPs.

Hence, all these data structures fail for use in electronic design automation either for reasons of space complexity (e.g. truth tables) or time complexity of the operations (e.g branching programs).

3 OBDDs – Ordered Binary Decision Diagrams

In 1986, by introducing *ordered binary decision diagrams (OBDDs)*, Randall E. Bryant got ahead a fundamental step in the search for suitable data structures in circuit design [4,6]. In contrast to conventional descriptions based on computation rules, OBDDs are based on a decision process. That way, Bryant's OBDDs combine two advantages: the new established data structure is not only quite space efficient but can also be handled efficiently from the algorithmic point of view.

3.1 Definitions

Definition 1. An Ordered Binary Decision Diagram (OBDD) P for a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is a directed acyclic graph consisting of inner nodes labeled by Boolean variables and sinks labeled by the Boolean constants 1 and 0. Each inner node has two outgoing edges: the 1-edge and the 0-edge. The OBDD has a starting node called root. The computation of $f(a_1, \dots, a_n)$ follows a path from the root to a sink, where on a node labeled by x_i the input bit a_i is tested. If $a_i = 1$, the path follows the 1-edge, otherwise the 0-edge. The value of the reached sink determines the value of $f(a_1, \dots, a_n)$. On a path from the root to the sink, each variable occurs at most once. The variables on a path respect a given order, which is (possibly after renaming) x_1, \dots, x_n . For an edge leading from a node labeled by x_i to a node labeled by x_j it follows that $j > i$.

Figure 1 gives two examples for OBDDs for the Boolean function $f = bc + a\bar{b}\bar{c}$ w.r.t the variable order $a < b < c$.

From the Shannon decomposition one can derive the first important property of OBDDs:

Property 1 (Universality). Any Boolean function can be represented by an OBDD w.r.t any predefined variable order.

Some difficulties in handling Boolean functions in terms of decision diagrams are caused by the missing uniqueness, like in many other representations. By using a surprisingly simple reduction mechanism, for OBDDs this problem can be solved very elegantly. Obviously, the following two reduction rules keep the represented function invariant:

Elimination rule: If 1- and 0-edge of a node v point to the same node u , then eliminate v , and redirect all incoming edges to u .

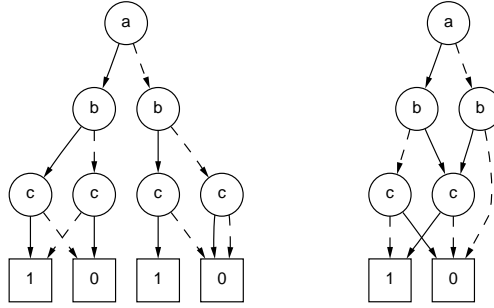


Fig. 1. Two OBDDs of $f = bc + a\bar{b}\bar{c}$

Merging rule: All terminal nodes with a given label are merged to one node, redirect all incoming edges to this node. If the non-terminal nodes u and v are labeled by the same variable, their 1-edges lead to the same node and their 0-edges lead to the same node, then eliminate one of the two nodes u, v , and redirect all incoming edges to the remaining node.

The elimination rule and the merging rule are illustrated in Figure 2.

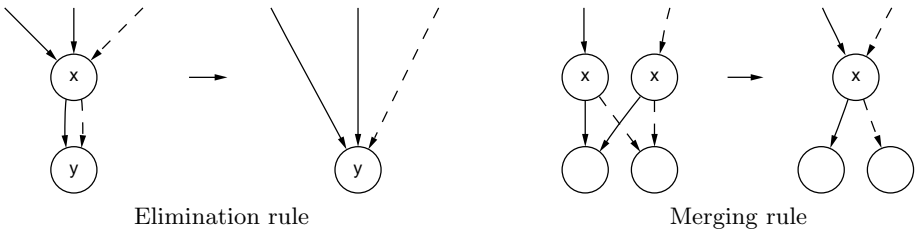


Fig. 2. Reduction rules

Definition 2. An OBDD is called *reduced* if none of the two reduction rules can be applied.

It is easy to see that the right OBDD in Figure 1 is reduced. Regarding the algorithmic properties of reduced OBDDs, the following property of canonicity is of basic importance:

Property 2 (Canonicity). With respect to a fixed variable order, the reduced OBDD of a Boolean function f is determined uniquely.

Besides universality and canonicity, OBDDs have third fundamental property, which makes OBDDs such a successful data structure for representation of Boolean functions: the efficiency in algorithmic manipulation.

3.2 Operations on and with OBDDs

OBDDs are the only data structure for the representation of switching functions that has deterministic polynomial algorithms for all of the above mentioned important operations. Operations on truth tables are input efficient too, but the size of the truth table is generally exponential in the number of variables. In the following the runtimes and space requirements for these operations are given ($\#P_f$ denotes the number of nodes in the OBDD P for the function f):

Satisfiability test: Runtime: $O(\#P_f)$

Equivalence test: Runtime: $O(\min(\#P_f, \#Q_g))$

Evaluation: Runtime: $O(n)$

Synthesis: Runtime and space: $O(\#P_f \cdot \#Q_g)$

Replacement by function: Runtime and space: $O((\#P_f)^2 \cdot \#Q_g)$

Minimization: (w.r.t a given order) Runtime and space: $O(\#P_f)$

It is worthwhile mentioning that most operations (except synthesis and replacement by function) have time and space requirements linear in the size of the OBDD. So, together with an efficient implementation OBDDs form a powerful data structure.

3.3 Efficient Synthesis of OBDDs

By \otimes we denote an arbitrary binary Boolean operation, e.g. the conjunction or the disjunction. In order to compute the OBDD of $f \otimes g$ from the OBDD representations of two functions f and g , one can use Shannon's decomposition w.r.t. the leading variable x in the variable order π :

$$f \otimes g = x (f|_{x=1} \otimes g|_{x=1}) + \bar{x} (f|_{x=0} \otimes g|_{x=0}),$$

where $f|_{x=1}$ is the subfunction that results from f after replacing the variable x by the constant 1. By repeated application of this decomposition an OBDD representation of the function $f \otimes g$ is computed. In order to perform this operation efficiently, multiple calls with the same argument pairs have to be avoided – instead, the already computed results from earlier stages are being recalled from a *computed-table*. In this way, the originally exponential number of decompositions is now bounded by the product of the two OBDD-sizes.

To increase the usage of the computed table all synthesis operations are mapped to a single operation, the so called if-then-else operator (ITE):

$$ITE(f, g, h) = f \cdot g + \bar{f} \cdot h.$$

E.g. $h = f \cdot g$ maps to $h = ITE(f, g, 0)$. Because of the huge number of ITE operations during synthesis the computed table is usually implemented as a cache to reduce memory consumption.

Another helpful construction is the usage of a *unique-table* which holds in a hash-table all already represented nodes. Before a new node is introduced its existence is checked in the unique-table. Together with an immediate check for the elimination rule the constructed OBDDs are always reduced and the reduction operation becomes obsolete.

3.4 Construction of OBDDs: Symbolic Simulation

The process of constructing an OBDD representations for a given circuit is called *symbolic simulation* of this circuit. Symbolic simulation is based on the iterated application of the synthesis operation:

Starting with the (trivial) OBDD representations of the input nodes one constructs, in topological order, OBDDs for each gate from the OBDDs of the corresponding predecessor gates applying the synthesis operation corresponding to the operation of the gate.

Of course, it may happen that the OBDDs of the circuits are quite large. However, many circuits of the real world inherently contain much structure – hence, the reduction rules of the OBDDs cause the graphs describing the circuit to remain small.

3.5 Influence of the Variable Order on the OBDD Size

The size of an OBDD and hence the complexity of its manipulation heavily depends on the underlying variable order. An example is shown in Figure 3. With respect to the variable order $x_1, x_2, \dots, x_{2n-1}, x_{2n}$ the function

$$x_1x_2 + x_3x_4 + \dots + x_{2n-1}x_{2n}$$

has an OBDD representation of linear size. For the variable order $x_1, x_3, \dots, x_{2n-1}, x_2, x_4, \dots, x_{2n}$ however, the size of the OBDD grows exponentially in n .

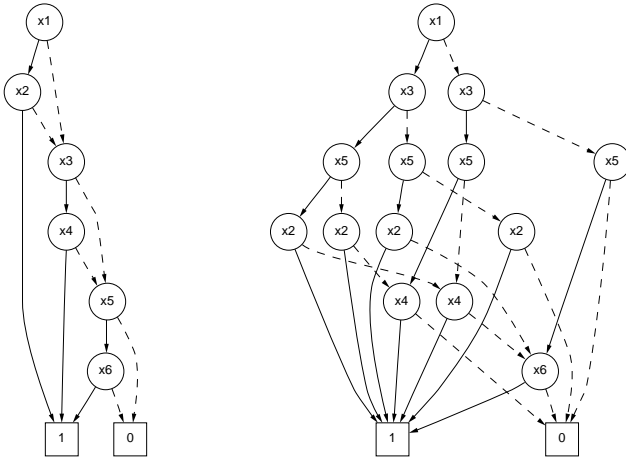


Fig. 3. Influence of the variable order

The same effect occurs in the case of adder functions: Depending on the variable order, the OBDD-size varies from linear to exponential in the number

of input bits. Other important functions, e.g. the multiplication of two n -bit numbers imply OBDDs of exponential size w.r.t. every variable order [5].

Due to the uniqueness of the OBDD representation w.r.t a given variable order, finding a suited variable order is the only way to optimize the size of the OBDD representation of a Boolean function.

In Section 5 we will discuss this problem in more detail.

4 Paradigmatic Applications of OBDDs in Formal Verification

In the following we will discuss three paradigmatic applications of OBDDs: (1) Formal verification of combinatorial circuits, (2) analysis of sequential systems and (3) symbolic model checking.

4.1 Verification of Combinatorial Circuits

The task in verification of combinatorial circuits is to check whether an implementation of a circuit C fulfills its specification S , i.e. both S and C produce the same functional outputs for all inputs. It can be seen that this is an intractable problem if all inputs variations to the circuit are tested explicitly.

OBDD based combinatorial verification proceeds in two steps:

1. Construct the OBDDs P_C and P_S for C and S by symbolic simulation.
2. Check the equivalence of the OBDDs P_C and P_S .

In case of a strong canonical representation, i.e when both functions are represented within the same OBDD, the equivalence test itself consists of a single pointer comparison. Each step in the iteration can be performed efficiently w.r.t. the OBDD-sizes of the predecessor gates. This shows that the difficulty of the NP-complete equivalent test [14] has now been shifted into the representation size.

If C and S are not equivalent the operation $P_C \oplus P_S$ gives the inputs where implementation and specification differ. This may be used for debugging.

4.2 Formal Verification of Sequential Systems

Equivalence Check for Two Finite State Machines

A central task in formal verification of sequential systems like controllers or protocols is the test for equivalence of two given finite state machines (FSMs). This is needed if correctness of an FSM has to be checked after an optimization process.

The equivalence test of two finite state machines M_1 and M_2 itself can be reduced to a reachability analysis by using the construction in Figure 4: Let M denote the so-called product machine whose state space is the Cartesian product of the state spaces of M_1 and M_2 . The output of M for a given state and a given

input is 1 if and only if for this configuration the outputs of M_1 and M_2 agree. M_1 and M_2 have the same input/output behavior if and only if the output of M is 1 for all reachable states. Hence, reachability analysis is the key problem

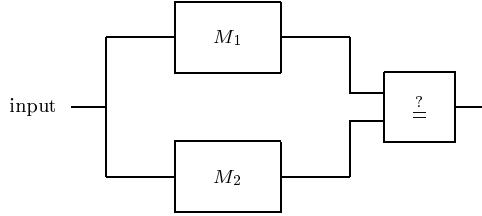


Fig. 4. Product machine for FSM verification

Reachability Analysis

The increasing complexity of sequential systems requires efficient techniques to be able to perform reachability analysis.

BFS Traversal

Since the set of reachable states can be quite large, an explicit representation of this set, e.g. in form of a list, cannot be suitable under any circumstances. Coudert, Berthet and Madre have investigated the characteristic function of state sets which can be considered as a Boolean function and therefore be represented by an OBDD [7,9]. They have shown that this representation form goes well together with the operations which have to be performed for the computation of the reachable states: If reachable states are computed according to a breadth-first-traversal then the representation via the characteristic function allows to compute all corresponding successor states within a single computation. For this reason, one also uses the term *symbolic breadth-first traversal*. Once more, the complexity of the computation depends on the OBDD-size of the occurring state sets.

Figure 5 gives an outline of the algorithm.

Image Computation

The computation of the reachable states is a core task for optimization and verification of sequential systems. The essential part of BDD-based traversal techniques is the transition relation (TR):

$$\text{TR}(x, y) = \prod_i \delta_i(x_i) \equiv y_i,$$

which is the conjunction of the transition relations of all latches (δ_i denotes the transition function of the i th latch).

```

traverse( $\delta, q_0$ ) {
  /* Input: Next-state function  $\delta$ ,
  initial set  $S_0$  */
  /* Output: Set of reachable states
  */
  Reached = From =  $S_0$ ;
  Do {
    To =  $\text{Im}(\delta, \text{From})$ ;
    New = To \ Reached;
    From = New;

    Reached = Reached  $\cup$  New;
  } While (New  $\neq \emptyset$ );
  Return Reached;
}

```

Fig. 5. Basic algorithm for reachability analysis based on breadth-first traversal

Partitioned Transition Relation

The transition relation is *monolithically* represented as a single OBDD and such a monolithic representation is usually much too large to allow an efficient computation of the reachable states. Therefore, more sophisticated reachable states computation methods make use of a *partitioned* TR [1], i.e. a cluster of OBDDs each of them representing the TR of a subgroup of latches. A transition relation partitioned over sets of latches L_1, \dots, L_j can be described as follows:

$$\text{TR}(x, y) = \prod_j \prod_{i \in L_j} \delta_i(x_i) \equiv y_i.$$

The RS computation consists of repeated image computations $\text{Img}(\text{TR}, R)$ of a set of already reached states R :

$$\text{Img}(\text{TR}, R) = \exists_x (\text{TR}(x, y) \cdot R)$$

With the use of a partitioned TR the image computation can be iterated over L_j and the \exists operation can be applied during the product computation (*early quantification*).

Early Quantification

The so called *AndExist* [1] or *AndAbstract* operation ($\exists_x (f \cdot g)$) may prevent a blow-up of BDD-size during the image computation. Another important problem is finding an optimal schedule of the partitions for the AndExist operation. Geist and Beer [15] presented a heuristic for the ordering of partitions each representing a single state variable.

OBDD Size Explosion During Image Computation

Even with the use of partitioned transition relation and early quantification the size for intermediate OBDDs and the OBDDs representing the reachable state set tend to behave as shown in Figure 6. This problem of explosion of memory consumption is a major problem of current research.

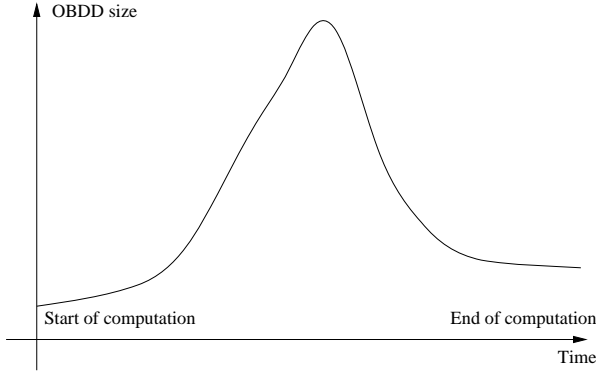


Fig. 6. Memory consumption during image computation

4.3 Symbolic Model Checking

Since a complete formal verification of a sequential system is often too complex, methods are of interest that guarantee at least correctness of certain properties. One of them is the so-called *model checking*.

Model checking is the problem to decide whether an implementation satisfies its specification given in terms of a temporal logic, e.g., the so-called *computation tree logic* (CTL). The idea to combine model checking with the symbolic OBDD algorithms was first introduced by McMillan [24] and Coudert/Madre [8]. Using this way of *symbolic* model checking, real-life systems up to 10^{100} states can be verified.

The formulas of CTL describe properties of infinite pathes of states that are traversed during the computation.

CTL and Fairness Constraints

The formulas of CTL describe properties of infinite pathes of states that are traversed during the computation. In addition to the Boolean operators \wedge , \vee and \neg , CTL has four temporal operators:

- X** The next operator describes a condition that is true in the next state of the computation.

- G** The global operator describes a condition that is true for all states of a path.
- F** The future operator describes a condition that is true on a path sometimes in the future.
- U** The until operator $a\mathbf{U}b$ is true on a path if a is true until b is true.

All temporal operators are quantified with either **A** (on all pathes it holds. . .) or **E** (there exists a path, where. . .).

The **X** operator can be described in terms of the transition relation and Boolean operations. The other operators are computed by fixpoint iterations of the **X** operator.

5 Optimization of Variable Ordering for OBDDs

Due to the strong dependence of the OBDD-size upon the chosen variable order it is one of the most important problems in the use of OBDDs to construct “good” orders, i.e. orders that fit well to the represented function. However, the problem to construct an optimal order of a given OBDD is known to be NP-hard [41,3]. The currently best known exact procedure is based on dynamic programming and has running time $O(n^2 \cdot 3^n)$ [13]. Unfortunately, for real-life applications this method is useless. To make the problem even worse, Sieling [39] has shown that there is no polynomial time approximation scheme for the variable ordering problem unless $P=NP$.

5.1 Static Techniques

There exist a variety of heuristics to determine a variable ordering before building the OBDD of a function. These heuristics utilize various informations given by the netlist of the function [23,29]. It turned out that these heuristics often work only for very specific functions and are not suitable for the case when the functions change during the computation as it is often the case in e.g. reachability analysis.

Nevertheless these heuristics might be useful to determine starting orders.

5.2 Dynamic Techniques

Dynamic Variable Reordering is the process of improving the variable order and hence the size of an already built OBDD. Virtually, any optimization paradigm has been applied to variable reordering from genetic techniques to simulated annealing but one of the most successful strategies still is the local search algorithm proposed by Ruddell 1993. The so called *Sifting* [37] algorithm is based on the swap operation of two variables in the order, which can be carried out locally and hence is very efficient.

In the following we describe some state-of-the-art dynamic variable reordering techniques based on the sifting algorithm:

Group Sifting In this method mutually attracting variables are united into a group. Then, within the sifting algorithm instead of single variables the whole group is moved through the order.. There are two different possible sifting based reordering strategies:

1. The individual blocks are reordered by means of a sifting strategy.
2. The variables within each block are reordered by means of a sifting strategy.

Recently, a method called *lazy group sifting* [18] has been introduced. This method relaxes the grouping of present and next state variables during the process of reachable states computation.

Lower Bound Sifting The *lower bound sifting* algorithm[10] utilizes the theoretical lower bounds of the swap operation [3] to determine when to stop sifting of a variable if there is no improvement in size possible.

Linear Sifting The *linear sifting* algorithm [33,34] combines the efficiency of Sifting and the power of linear transformations. Linear transformations replace one variable x_i by a linear combination of this variable with another. A linear combination is obtained by taking the exclusive or of the arguments or its complement. Linear transformations considerably enlarge the search space for the OBDD optimization, when compared to variable reordering, still they enjoy efficient manipulation.

Block Restricted Sifting The idea behind this method [25] is to move the variables during reordering only within fixed blocks instead of moving them through the complete order. From theory it is known, that changing the variable order of a block does not affect the size of the other blocks. The determination of the block boundaries follows from a communication complexity argument. A small information flow between two parts of an OBDD indicates a good candidate for a block boundary. If there is only little information flow between two blocks the distribution of variables to these blocks is well chosen. Improving the variable order inside the blocks might lead to a significant reduction of the OBDD size. The information flow is best indicated by the number of subfunctions that cross one level.

Sample Sifting Sampling is a common heuristic technique applied to optimization problems with huge search spaces. The idea behind the sampling strategy is to choose a relevant sample from the given problem instance to solve the optimization problem for the chosen subset and to generalize the solution to the complete instance.

Applied to the problem of reordering OBDD variables the sampling strategy can be described as follows [40,19]:

1. Choose some OBDDs or subOBDDs from the common shared OBDD.
2. Copy these OBDDs to a different location.
3. Reorder only the Sample.
4. Shuffle the variables of the original BDD to the newly computed order of the sample.

In sequential verification the time needed for reordering often is unacceptably long. The lazy group sifting algorithm and adaptations of block restricted sifting and sample sifting [32,35] address this problem.

6 Variants of the BDD Data Structure

Another idea to improve the applicability of the BDD data structure is to introduce more powerful concepts.

Here we will discuss some promising approaches that are important research topics.

6.1 Ordered Functional Decision Diagrams (OFDDs)

If f denotes the function being represented by an OBDD-node with label x_i , and g, h denote the functions being represented in the two sons, then Shannon's decomposition holds:

$$f = x_i g + \overline{x_i} h.$$

However, it is also possible to perform other decompositions in the nodes, e.g. the so-called Reed-Muller decomposition

$$f = g \oplus x_i h.$$

These *ordered functional decision diagrams* (OFDDs), introduced in [21], are particularly suited in the context of problems based on the exclusive-or operation, e.g. the minimization of AND-XOR-polynomials. One step further, in [12] it is shown that different decomposition types can be combined within the same subgraph while preserving good algorithmic properties (*ordered Kronecker functional decision diagrams*, OKFDDs).

6.2 Transformed Binary Decision Diagrams (TBDDs)

Another variant of the OBDD concept converts Boolean functions into functions with easier representations [2], similar to classic transformation concepts like Fourier transformation.

More precisely, let \mathbb{B}_n denote the set of all n -variable Boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$. The transformation approach is based on *cube transformations* τ which are bijective mappings from $\{0, 1\}^n \rightarrow \{0, 1\}^n$. A cube transformation τ induces a mapping $\Phi_\tau : \mathbb{B}_n \rightarrow \mathbb{B}_n$ with $\Phi_\tau(f)(a) = f(\tau(a))$ for every $a = (a_1, \dots, a_n) \in \{0, 1\}^n$. Now, instead of representing and manipulating the original function $f \in \mathbb{B}_n$, the idea is to use the transformed function $\Phi_\tau(f) = f(\tau)$. This variable transformation preserves the efficient manipulation as shown by the following fact:

Proposition 1. *If $\tau : \{0, 1\}^n \rightarrow \{0, 1\}^n$ is a cube transformation, and $f_1, f_2 \in \mathbb{B}_n$ are Boolean functions, then Φ_τ defines an automorphism on \mathbb{B}_n , i.e. the following holds:*

1. $f_1 = g_1$ if and only if $\Phi_\tau(f_1) = \Phi_\tau(g_1)$.
2. Let \otimes be any binary operation on \mathbb{B}_n . If $f = f_1 \otimes f_2$, then $\Phi_\tau(f) = \Phi_\tau(f_1) \otimes \Phi_\tau(f_2)$.

In other words, due to the second statement the polynomial complexity of the Boolean operations remain valid even if we work with the transformed functions. If, for example, one wants to check two given functions for equivalence, statement 1 tells us, that in this situation it is not necessary at all to re-transform the functions.

The realization of this general framework requires to find good techniques for finding suitable cube transformations τ which lead to small OBDD-sizes for the transformed versions of the relevant functions. Suitable transformations that have already demonstrated their optimization power include graph-driven transformations based on complete types [2] and linear transformations [27,26].

6.3 Parity-OBDDs (\oplus -OBDDs)

Besides relaxing the ordering restriction [16] or the usage of different decomposition types for Boolean functions, we are focusing on the extension of OBDDs with functional operator nodes, esp. \oplus -OBDDs (Mod2-OBDDs), i.e. OBDDs with additional operator nodes computing the Boolean parity of their successors [17]. By introducing parity nodes the representation has the potential of being more compact while on the other hand giving up canonicity. More precisely, a \oplus -OBDD P over a set $X_n = \{x_1, \dots, x_n\}$ of Boolean variables is a rooted directed acyclic connected graph $P = (V, E)$. V is the set of nodes, consisting of non-terminal nodes with out-degree 2 and of terminal nodes with out-degree 0. The two terminal nodes with no outgoing arcs are labeled by the Boolean constants 0 and 1. The remaining nodes are either labeled with Boolean variables $x_i \in X_n$ (*branching nodes*), or with the binary Boolean function \oplus (EXOR) (\oplus -nodes, *functional nodes*). On each path, every variable is permitted to occur at most once.

The function f_P associated with the \oplus -OBDD P is determined in the following way: For a given input assignment $a = (a_1, \dots, a_n) \in \{0, 1\}^n$, the function f_P associated with the \oplus -OBDD P is determined by extending the Boolean values assigned to the leaves to all other nodes of P as follows:

- Let v_0 and v_1 be the successors of v , carrying the Boolean values $\delta_0, \delta_1 \in \{0, 1\}$.
- If v is a branching node, $l(v) = x_i \in X_n$, then v is associated with δ_{a_i} .
- If v is a \oplus -node, then v is associated with $\oplus(\delta_0, \delta_1) = (\delta_0 + \delta_1) \bmod 2$.

The function $f_P(a)$ takes the value associated with the source of P .

\oplus -OBDD are not a canonical representation for Boolean functions, i.e. there can be different distinct \oplus -OBDD representations for the same Boolean function. Therefore, the identification of two \oplus -OBDDs representing the same Boolean function becomes an essential operation.

The fastest known deterministic equivalence test for \oplus -OBDDs requires time cubic in the number of nodes [42], and therefore, it is not suited for an application within a practical working environment. Equivalence of \oplus -OBDDs can also be tested probabilistically [17]. The \oplus -OBDDs are algebraically transformed to

polynomials over a Galois field and finally the equivalence of two polynomials is tested [20]. Thus, based on an efficient implementation of this technique, the probabilistic equivalence test can be performed in constant time [30].

For the minimization of \oplus -OBDDs not only the variable order has to be taken under consideration. Also the number and the placement of the \oplus -nodes within the \oplus -OBDD has a major influence on the size [30]. Therefore, sophisticated heuristics including all these requirements based on an efficient implementation, ranging from the exchange of adjacent variables to implanting and moving \oplus -nodes have to be developed [31].

7 Establishing a WWW Portal for BDD Research

Today the Internet offers the possibility of standardized and global communication without a need for special hardware or expensive infrastructure. While a lot of resources are available on the World Wide Web (WWW), there is no central place to access these resources in a convenient way. There are search engines, but they are indexing only small segments of the web. A recent survey [22] had the search engine with the best coverage at only 13% of all pages. Even where search engines suffice, the results of their searches are unstructured and often the user is presented with an all-or-nothing situation, where queries either return far too many hits or none at all. One solution is the recent notion of so called *portal*-sites, that organize WWW contents into categories and in some cases even grade the quality. But what efforts are underway are mostly targeted at the general public and relatively small communities, like research communities, are not commercially interesting to the existing portals.

To address the needs of the Decision Diagram research community, we have created a specialized portal site (www.bdd-portal.org) [36] that provides easy access to researchers, events, literature and other material in the area of Decision Diagrams. Furthermore this portal includes a system for providing online decision diagram functionality by allowing the use of heuristics and tools via a WWW interface. The system is mainly aimed at facilitating the process of determining which heuristic is best suitable for a special application and at allowing easy comparison between heuristics. It is open to all researchers who wish to publicize their results in this way. As applications using Decision Diagrams are usually quite memory and CPU intensive, the system represents a major effort and greatly facilitates the evaluation of research results in Decision Diagram heuristics. Given the importance of OBDDs and other types of Decision Diagrams in verification, simulation, synthesis and similar uses, such an effort seems to have been overdue.

References

1. J. R. Burch, E. M. Clarke, D. E. Long, *Symbolic Model Checking with partitioned transition relations*, Proc. of Int. Conf. on VLSI, 1991.

2. J. Bern, Ch. Meinel, and A. Slobodová. OBDD-based Boolean manipulation in CAD beyond current limits. In *Proc. 32nd ACM/IEEE Design Automation Conference (San Francisco, CA)*, pages 408–413, 1995.
3. B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45:993–1002, 1996.
4. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.
5. R. E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, C-40:205–213, 1991.
6. R. E. Bryant. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
7. O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines using symbolic execution. In *Proc. Workshop on Automatic Verification Methods for Finite State Machines*, volume 407 of *Lecture Notes in Computer Science*, pages 365–373. Springer, 1989.
8. O. Coudert and J. C. Madre. A unified framework for the formal verification of sequential circuits. In *Proc. IEEE International Conference on Computer-Aided Design*, pages 126–129, 1990.
9. O. Coudert and J. C. Madre. The implicit set paradigm: A new approach to finite state system verification. *Formal Methods in System Design*, 6(2):133–145, 1995.
10. R. Drechsler, W. Günther, Using Lower Bounds During Dynamic BDD Minimization, in *Proc. of IEEE International Conference on Computer-Aided Design*, pages 29–32, 1999.
11. E. Dubrova, H. Sack, Probabilistic verification of multiple-valued functions, *Proc. of the 30th Int. Symp. on Multiple Valued Logic (ISMVL2000)*, 2000.
12. R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In *Proc. 31st ACM/IEEE Design Automation Conference (San Diego, CA)*, pages 415–419, 1994.
13. S. J. Friedman and K. J. Supowit. Finding the optimal variable ordering for binary decision diagrams. *IEEE Transactions on Computers*, 39:710–713, 1990.
14. M. R. Garey and M. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1978.
15. D. Geist and I. Beer. Efficient model checking by automated ordering of transition relation partitions. In *Proc. Computer-Aided Verification*, volume 818, pages 299–310, 1994.
16. J. Gergov and Ch. Meinel. Frontiers of feasible and probabilistic feasible Boolean manipulation with branching programs. In *Symposium on Theoretical Aspects in Computer Science*, volume 665 of *Lecture Notes in Computer Science*, pages 576–585. Springer, 1993.
17. J. Gergov and Ch. Meinel. MOD-2-OBDDs - a data structure that generalizes EXOR-sum-of-products and ordered binary decision diagrams. *Formal Methods in System Design*, 8:273–282, 1996.
18. H. Higuchi, F. Somenzi, Lazy Group Sifting for Efficient Symbolic State Traversal of FSMs, in *Proc. of IEEE International Conference on Computer-Aided Design*, 1999.
19. J. Jain, W. Adams and M. Fujita, Sampling schemes for computing OBDD variable orderings, *Proc. IEEE International Conference on Computer-Aided Design*, pages 331–638, 1998.

20. J. Jain, J. Bitner, D. S. Fussell, J. A. Abraham, Probabilistic verification of Boolean functions, in *Formal Methods in System Design 1*, Kluwer Academic Publishers, pages 65-116, 1992.
21. U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *Proc. European Design Automation Conference*, pages 43-47, 1992.
22. S. Lawrence and C.L. Gilles, Accessibility of information on the web, *Nature*, Volume 400, Number 6740, 1999.
23. S. Malik, A. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Proc. IEEE International Conference on Computer-Aided Design (Santa Clara, CA)*, pages 6-9, 1988.
24. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
25. C. Meinel and A. Slobodová, Speeding up Variable Reordering of OBDDs, in *Proc. of the International Conference on Computer Design*, pages 338-343, 1997.
26. Ch. Meinel, F. Somenzi, and T. Theobald. Linear sifting of decision diagrams. In *Proc. 34th ACM/IEEE Design Automation Conference (Anaheim, CA)*, pages 202-207, 1997.
27. Ch. Meinel and T. Theobald. Local encoding transformations for optimizing OBDD-representations of finite state machines. In *Proc. International Conference on Formal Methods in Computer-Aided Design (Palo Alto, CA)*, volume 1166 of *Lecture Notes in Computer Science*, pages 404-418. Springer, 1996.
28. C. Meinel and T. Theobald, *Algorithms and datastructures in VLSI-Design*, Springer, 1998.
29. S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagrams with attributed edges. In *Proc. 27th ACM/IEEE Design Automation Conference (Florida, FL)*, pages 52-57, 1990.
30. C. Meinel, H. Sack, \oplus -OBDDs - a BDD structure for probabilistic verification, in *Proc. Workshop on Probabilistic Methods in Verification* pages 141-151, 1998.
31. Ch. Meinel, H. Sack, Algorithmic Considerations for Parity-OBDD Reordering, in *Proc. of the 1999 IEEE/ACM Int. Workshop on Logic Synthesis (IWLS99)*, 1999.
32. Ch. Meinel, K. Schwetmann, A. Slobodová, Application Driven Variable Reordering and an Example Implementation in Reachability Analysis, in *Proc. of ASP-DAC'99*, Hongkong, pages 327-330, 1999.
33. Ch. Meinel, F. Somenzi, T. Theobald Linear Sifting of Decision Diagrams in *Proc. of 34th IEEE International Conference on Computer-Aided Design*, Anaheim (USA), IEEE Computer Society Press, pages 202-207, 1997.
34. Ch. Meinel, F. Somenzi, T. Theobald, Function Decomposition and Synthesis Using Linear Sifting, in *Proc. ASP-DAC'98*, Yokohama (Japan), pages 81-86, 1998.
35. Ch. Meinel, Ch. Stangier, Speeding up Symbolic Model Checking by Accelerating Dynamic Variable Reordering , in *Proc. of 10th ACM Great Lake Symposium on VLSI*, Chicago (USA), pages 39-42, 2000.
36. Ch. Meinel and A. Wagner, WWW.BDD-Portal.Org - An Electronic Basis for Cooperative Research in EDA, *Proc. CRIS 2000*, Espoo, Helsinki, Finland, USA, 2000.
37. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. IEEE International Conference on Computer-Aided Design (Santa Clara, CA)*, pages 42-47, 1993.
38. H. Sack, E. Dubrova, Ch. Meinel Mod-p Decision Diagrams: A Data-Structure for Multiple-Valued Functions, in *Proc. of the 2000 IEEE/ACM Int. Workshop on Logic Synthesis (IWLS2000)*, 2000.

39. D. Sieling, On the Existence of Polynomial Time Approximation Schemes for OBDD Minimization. in *Proc. of Symposium on theoretical aspects in Computer Science*, pages 205–215, 1998.
40. A. Slobodová and C. Meinel, Sample Method for Minimization of OBDDs. In *Proc. of the International Workshop on Logic Synthesis*, pages 311–316, 1998.
41. S. Tani, K. Hamaguchi, and S. Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams. In *Proc. International Symposium on Algorithms and Computation '93*, volume 762 of *Lecture Notes in Computer Science*, pages 389–398. Springer, 1993.
42. S. Waack, On the descriptive and algorithmic power of parity ordered binary decision diagrams, *Proc. 14th Symp. on Theoretical Aspects of Computer Science*, 1200 of LNCS, Springer, 1997.

Scheduling under Uncertainty: Bounding the Makespan Distribution

Rolf H. Möhring*

Technische Universität Berlin, 10623 Berlin, Germany

moehring@math.tu-berlin.de,

<http://www.math.tu-berlin.de/~moehring/>

Abstract. Deterministic models for project scheduling and control suffer from the fact that they assume complete information and neglect random influences that occur during project execution. A typical consequence is the underestimation of the expected project duration and cost frequently observed in practice. This phenomenon occurs even in the absence of resource constraints, and has been the subject of extensive research in discrete mathematics and operations research.

This article presents a survey on the reasons for this phenomenon, its complexity, and on methods how to obtain more relevant information. To this end, we consider scheduling models with fixed precedence constraints, but (independent) random processing times. The objective then is to obtain information about the distribution of the project makespan. We will demonstrate that this is an $\#P$ -complete problem in general, and then consider several combinatorial methods to obtain approximate information about the makespan distribution.

1 Uncertainty in Scheduling

In real-life projects, it usually does not suffice to find good schedules for fixed deterministic processing times, since these times mostly are only rough estimates and subject to unpredictable changes due to unforeseen events such as weather conditions, obstruction of resource usage, delay of jobs and others.

In order to model such influences, the processing time of a job $j \in V$ is assumed to be a random variable \mathbf{p}_j . Then $\mathbf{p} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n)$ denotes the (random) vector of processing times, which is distributed according to a joint probability distribution Q . This distribution Q is assumed to be known, though sometimes, also partial information may suffice. In general, Q may contain stochastic dependencies, but the methods discussed in this paper require that the job processing times are stochastically independent (Methods for dependent processing times are quite different, see [14]). Furthermore, like in deterministic models, we have precedence constraints given by a directed acyclic graph $G = (V, E)$ but no resource constraints. (See [16] for an overview about the more involved case

* Supported by Deutsche Forschungsgemeinschaft under grant Mo 346/3-3 and by German Israeli Foundation under grant I-564-246.06/97

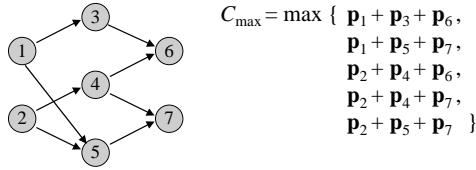


Fig. 1. An example project network and its makespan C_{\max} .

with resource constraints.) We refer to G also as the *project network* and to the pair (G, \mathbf{P}) as *stochastic project network*. In the literature, they are also referred to as PERT networks, since PERT was one of the first techniques to analyze the stochastic behavior of such networks.

Now consider a particular realization $p = (p_1, \dots, p_n)$ of the random processing time vector $\mathbf{p} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n)$. Since there are no resource constraints, every job j can complete at its earliest possible completion time $C_j = C_j(p)$, which is equal to the length of a longest path in G that ends with j , where the length of a job j is its processing time p_j .

The *earliest project completion* or *makespan* for the realization p is then $C_{\max}(p) := \max_j C_j(p) = \max_P \sum_{j \in P} p_j$, where P ranges over all inclusion-maximal paths of G . Since the processing times \mathbf{p}_j are random, the makespan C_{\max} is also a random variable, and it may be written as $C_{\max} = \max_P \sum_{j \in P} \mathbf{p}_j$, i.e., as the maximum of sums over subsets of a common set of random variables. An example is given in Figure 1.

Our objective is to obtain information about the distribution of this random variable C_{\max} .

The necessity to deal with uncertainty in project planning becomes obvious if one compares the “deterministic makespan” $C_{\max}(E(\mathbf{p}_1), \dots, E(\mathbf{p}_n))$ obtained from the expected processing times $E(\mathbf{p}_j)$ with the expected makespan $E(C_{\max}(\mathbf{p}))$. Even in the absence of resource constraints, there is a systematic underestimation $C_{\max}(E(\mathbf{p}_1), \dots, E(\mathbf{p}_n)) \leq E(C_{\max}(\mathbf{p}_1, \dots, \mathbf{p}_n))$ which may become arbitrarily large with increasing number of jobs or increasing variances of the processing times [9]. Equality holds if and only if there is one path that is the longest with probability 1, see Theorem 1 below. This systematic underestimation of the expected makespan has already been observed by Fulkeron [6]. The error becomes even worse if one compares the deterministic value $C_{\max}(E(\mathbf{p}_1), \dots, E(\mathbf{p}_n))$ with quantiles t_q such that $\text{Prob}\{C_{\max}(\mathbf{p}) \leq t_q\} \geq q$ for large values of q (say $q = 0.9$ or 0.95). A simple example is given in Figure 2 for a project with n parallel jobs that are independent and uniformly distributed on $[0, 2]$. Then the deterministic makespan $C_{\max}(E(\mathbf{p}_1), \dots, E(\mathbf{p}_n)) = 1$, while $\text{Prob}(C_{\max} \leq 1) \rightarrow 0$ for $n \rightarrow \infty$. Similarly, all quantiles $t_q \rightarrow 2$ for $n \rightarrow \infty$ (and $q > 0$).

This is the reason why good practical planning tools should incorporate stochastic methods.

We conclude this introduction with a proof of the underestimation error.

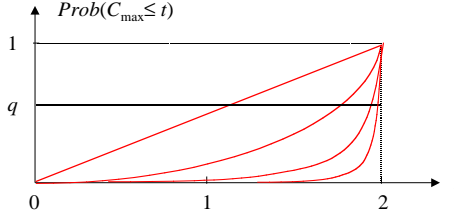


Fig. 2. Distribution function of the makespan for $n = 1, 2, 4, 8$ parallel jobs that are independent and uniformly distributed on $[0, 2]$.

Theorem 1. Let $G = (V, E)$ be a project network with random processing time vector \mathbf{p} . Then

$$C_{\max}(E(\mathbf{p}_1), \dots, E(\mathbf{p}_n)) \leq E(C_{\max}(\mathbf{p}_1, \dots, \mathbf{p}_n)).$$

Equality holds iff there is one path that is the longest with probability 1.

Proof. Since C_{\max} is the maximum of sums of processing times, it is obviously a convex function of p . Thus the inequality is a special case of Jensen's inequality for convex functions. We give here an elementary proof for C_{\max} .

Let P_1, \dots, P_k be the inclusion-maximal paths of G and let Y_1, \dots, Y_k denote their (random) length, i.e., $Y_i := \sum_{j \in P_i} \mathbf{p}_j$. Then $C_{\max} = \max_i Y_i$, and

$$\begin{aligned} C_{\max}(E(\mathbf{p})) &= \max_i \sum_{j \in P_i} E(\mathbf{p}_j) = \max_i E\left(\sum_{j \in P_i} \mathbf{p}_j\right) = \max_i E(Y_i) \\ &= E(Y_{i_0}) \quad \text{assume that the maximum is attained at } i_0 \\ &\leq E(\max_i Y_i) \quad \text{since } Y_{i_0} \leq \max_i Y_i \text{ as functions of } p \\ &= E(C_{\max}(\mathbf{p})). \end{aligned}$$

Now assume that Y_1 is the longest path with probability 1. Then, with probability 1, $C_{\max} = Y_1 \geq Y_i$. Hence $E(C_{\max}) = E(Y_1) \geq E(Y_i)$ and the above calculation yields $C_{\max}(E(\mathbf{p})) = \max_i E(Y_i) = E(Y_1) = E(C_{\max})$.

In the other direction assume that $E(C_{\max}(\mathbf{p})) = C_{\max}(E(\mathbf{p}))$. Let w.l.o.g. P_1 be the longest path w.r.t. expected processing times $E(\mathbf{p}_j)$. Then $E(Y_1) = E(C_{\max}(\mathbf{p}))$ and

$$\begin{aligned} 0 &= E(C_{\max}(\mathbf{p}) - C_{\max}(E(\mathbf{p}))) = E(\max_i Y_i - \max_i E(Y_i)) \\ &= E(\max_i E(Y_i) - Y_1) = \int (\max_i E(Y_i) - Y_1) dQ. \end{aligned}$$

Since the integrand is non-negative, it follows that it is 0 with probability 1. Hence $Y_1 = \max_i E(Y_i) = C_{\max}$ with probability 1. \square

2 The Complexity of Evaluating the Makespan Distribution

Due to the practical importance of stochastic scheduling, many methods have been developed over the last 35 years. For stochastically independent processing times, these methods can be roughly grouped into *simulation* methods, methods for *bounding or calculating the expected makespan*, methods for *analyzing the “most critical” path*, and methods for *bounding the whole distribution function of the makespan*. We refer to [1] for a general overview, and to [18] for an overview on bounding methods and methods for independent processing times.

Most of the early contributions have not been aware of the enormous inherent complexity of the problem, which was formally analyzed only 1988 by Hagstrom [8]. She considers the following two problems:

MEAN: Given a project network with discrete, independent processing times \mathbf{p}_j , compute the expected makespan $E(C_{\max}(\mathbf{p}))$.

DF: Given a project network with discrete, independent processing times \mathbf{p}_j and a time t , compute the probability $\text{Prob}\{C_{\max}(\mathbf{p}) \leq t\}$ that the project finishes by time t .

Hagstrom shows that DF and the 2-state versions of MEAN, in which every processing time \mathbf{p}_j has only two discrete values, are $\#\mathcal{P}$ -complete. (Any $\#\mathcal{P}$ -complete problem is polynomially equivalent to counting the number of Hamiltonian cycles of a graph and thus in particular \mathcal{NP} -complete). This result is derived from a fundamental result of Provan and Ball [20] on the $\#\mathcal{P}$ -completeness of reliability problems, see Theorem 2 below.

The complexity status of the general version of MEAN is open (only the 2-state version, which has a short encoding, is known to be $\#\mathcal{P}$ -complete). If the processing times \mathbf{p}_j may take more than 2 values, the problem has a longer encoding that in principle could admit a polynomial algorithm for solving MEAN. However, Hagstrom provides some evidence that problems with a long encoding may still be difficult, since MEAN and DF cannot be solved in time polynomial in the number of values of $C_{\max}(\mathbf{p})$ unless $\mathcal{P} = \mathcal{NP}$.

These results show that efficient methods for calculating the expected makespan or quantiles of the distribution function of the makespan are very unlikely to exist, and thus (although in retrospect) justify the great interest in approximate methods such as bounds, simulation etc.

We will show the $\#\mathcal{P}$ -completeness of the 2-state version of MEAN below in Theorem 2. To this end, we make use of a construction that envisions the jobs of our project as arcs of a directed graphs instead of vertices. This construction will also be useful for the bounding algorithms below.

This construction uses an *s, t-dag*, i.e., a directed acyclic graph $D = (N, A)$ with a unique source s and a unique sink t . Every job j of G is represented by an arc of D such that precedence constraints are preserved, i.e., if (i, j) is an edge of G , then there is a path from the end node of i to the start node of j in D . Figure 3 gives an example. Such a representation is called an *arc diagram*

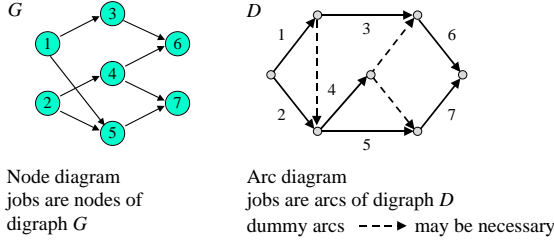


Fig. 3. Arc diagram of the project network of Figure 1.

(sometimes also *PERT network*) of the project. In general, one needs additional arcs (so-called *dummy arcs*) to properly represent the precedence constraints. A simple way to obtain an arc diagram is to take the original edges of G as dummy arcs, blow up every vertex of G to an arc with the same incoming and outgoing edges as in G , identify sources and sinks, and contract dummy arcs that are “superfluous” (i.e., the contraction does not change the original precedence constraints given by G). The arc diagram of Figure 3 is obtained in this way. Of course, arc diagrams are not unique, and it is \mathcal{NP} -hard to construct an arc diagram with the fewest possible dummy arcs [11]. For our purposes, the given polynomial construction suffices.

Theorem 2. [8] *The 2-state version of DF is $\#\mathcal{P}$ -complete.*

Proof. We will use the fact that the problem

RELIABILITY: Given an s, t -dag D with arc failure probabilities q_j (the *reliability network*), determine the probability that the s and t are joined by a path without arc failures (the *reliability*).

is $\#\mathcal{P}$ -complete [20]. From a given instance I of RELIABILITY, we will construct in polynomial time an instance I' of the 2-state version of DF such that the reliability of I equals 1 minus the probability that $C_{\max} \leq L - 1$ for some appropriately chosen value L . This obviously proves the theorem.

First, we take the s, t -dag D of instance I as arc diagram of instance I' . Every arc (job) j may attain two processing times, 0 with the failure probability q_j , and $p_j > 0$ with probability $1 - q_j$. The p_j are chosen in such a way that all paths from the source to the sink in D have the same length L . This may be obtained by starting with $p_j = 1$ for all j and then iteratively enlarging the p_j of those arcs j that are not yet on a path with the maximum length L until one path through j reaches this length L . Clearly, this is a polynomial algorithm that will output integer processing times $p_j \leq n$ such that all paths have length L . Then

$$\begin{aligned}
 \text{Prob}(\text{some path is reliable}) &= 1 - \text{Prob}(\text{all paths fail}) \\
 &= 1 - \text{Prob}(\cap_i \{\text{path } P_i \text{ fails}\}) \\
 &= 1 - \text{Prob}(\cap_i \{\text{path } P_i \text{ has length} < L\}) \\
 &= 1 - \text{Prob}(C_{\max} \leq L - 1). \quad \square
 \end{aligned}$$

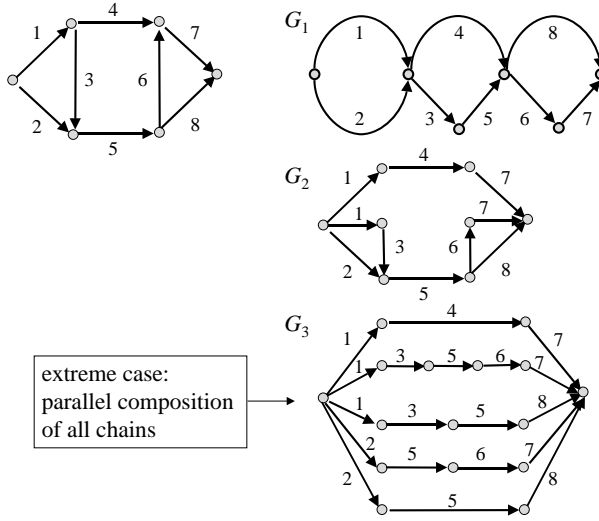


Fig. 4. Different networks G_i containing G as a minor in the arc diagram representation. Copies of a job have the same number.

3 A General Bounding Principle

Many of the methods that provide bounds for the distribution function of the makespan transform the given network (mostly represented as an arc diagram) into a network that is easier to evaluate. Typical examples in this respect are the bounds by Dodin [5], Kleindorfer [10], Shogan [22] and Spelde [23], see Section 5.

Möhring and Müller [17] give a unified model for such bounding results in terms of a *chain-minor* notion for project networks.

A network $G_1 = (V_1, E_1)$ is a *chain-minor* of a network $G_2 = (V_2, E_2)$ if (1) and (2) below hold.

- (1) Every job $j \in V_1$ is represented in G_2 by a set of *copies* or *duplicates* $D(h)$, where $D(h) \cap D(j) = \emptyset$ if $h \neq j$ and $\cup_j D(j) = V_2$.
- (2) Every chain C (the set of jobs on a path) of G_1 is “contained” in a chain C' of G_2 in the sense that, for every job $j \in C$, there is a duplicate $j' \in D(j)$ with $j' \in C'$. (These duplicates j may be different for different chains C of G_1 .)

Figure 4 gives an example.

Theorem 3. [17] *Let G be a chain-minor of H . Then one obtains a lower bound for the distribution function F_G of the makespan of G if one gives every duplicate j' of a job j the same processing time distribution as job j , treats them as independent, and calculates the distribution function F_H of the makespan of H . In other words,*

$$\text{Prob}\{C_{\max} \leq t \text{ in } G\} \geq \text{Prob}\{C_{\max} \leq t \text{ in } H\} \text{ for every } t.$$

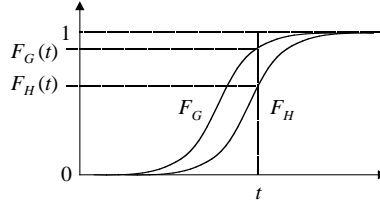


Fig. 5. An illustration of Theorem 3.

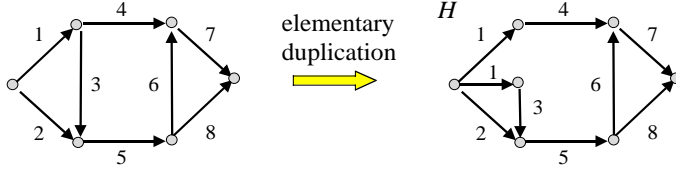


Fig. 6. The example networks for the proof of Theorem 3.

Figure 5 illustrates the relationship of F_G and F_H . Since $\text{Prob}\{C_{\max} \leq t \text{ in } G\} \geq \text{Prob}\{C_{\max} \leq t \text{ in } H\}$, F_H is called *stochastically greater* than F_G .

Proof. Let C_{\max}^G and C_{\max}^H denote the makespan in G and H , respectively. For a fixed realization p of processing times, the chain-minor property obviously ensures that $C_{\max}^G(p) \leq C_{\max}^H(p)$. It turns out that the independent variation of the processing time on duplicates of a job increases the makespan stochastically, i.e., $F_G \geq F_H$.

We will show this only for the special case of an “elementary duplication”, i.e., one job is duplicated into two copies. It can be shown that if G is a chain-minor of H , then there is a finite sequence $\mathcal{P}_0, \dots, \mathcal{P}_k$ of set systems such that \mathcal{P}_0 is the system $\mathcal{P}(G)$ of maximal paths of G , \mathcal{P}_k is the system $\mathcal{P}(H)$ of maximal paths of H , and any two neighbored set systems $\mathcal{P}_i, \mathcal{P}_{i+1}$ differ by either an *elementary duplication* (i.e., only one job is duplicated, and in only two copies) or a *proper inclusion* without any duplication as G and G_1 in Figure 4. However, the intermediate set systems need not be systems of paths of a project network anymore, but only systems of pairwise incomparable subsets of V . This does, however, not matter for the proof.

The bounding property in the case of a proper inclusion is straightforward. So suppose that G and H differ by an elementary duplication as in Figure 6. The proof for this example will be generic for arbitrary elementary duplications. It follows an argument given by Dodin [5] for proving his bound, see Section 5.1.

We must show that $\text{Prob}(C_{\max}^G \leq t) \geq \text{Prob}(C_{\max}^H \leq t)$ for every $t \geq 0$. Because of the stochastic independence of the processing times, both probabilities are measures in product spaces and can thus be computed using Fubini’s theorem. Although both spaces are different, they share the last components belonging to (p_2, \dots, p_8) . So both measures are obtained by measuring the “shadow”

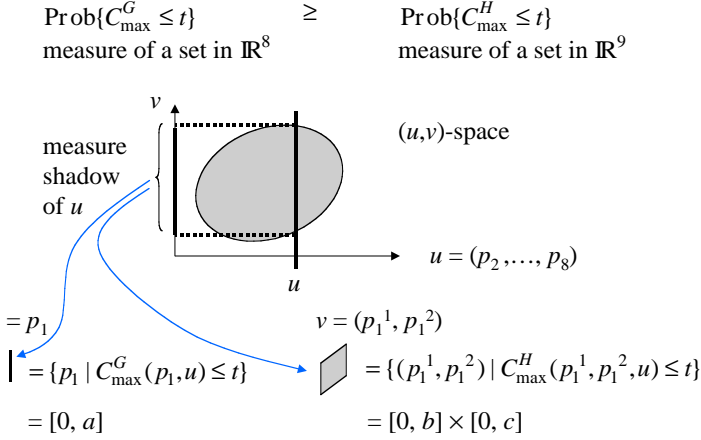


Fig. 7. Calculating probabilities by Fubini's theorem.

obtained by fixing (p_2, \dots, p_8) and integrating these shadows over (p_2, \dots, p_8) . This is illustrated by Figure 7.

So it suffices to show that $Q_1(\{p_1 \mid C_{\max}^G(p_1, u) \geq t\}) \leq Q_1 \otimes Q_1(\{(p_1^1, p_1^2) \mid C_{\max}^G(p_1^1, p_1^2, u) \geq t\})$. Here Q_1 is the distribution of \mathbf{p}_1 , and $Q_1 \otimes Q_1(\{\dots\})$ measures the independent variation of \mathbf{p}_1 on the two copies of job 1 in H for fixed u . Let $A := \{p_1 \mid C_{\max}^G(p_1, u) \geq t\}$ and $B := \{(p_1^1, p_1^2) \mid C_{\max}^G(p_1^1, p_1^2, u) \geq t\}$. Obviously, A is a 1-dimensional interval, say $A = [0, a]$. Similarly, B is a 2-dimensional interval, say $B = [0, b] \times [0, c]$, since the two copies of job 1 can independently achieve their respective maxima b and c .

Assume w.l.o.g. that $b \geq c$. Then $a \geq c$. If not, then $a < c$, and the processing time vector (c, u) would give a makespan $C_{\max}^G(c, u) > t$ in G , while (c, c, u) would yield a makespan $C_{\max}^H(c, c, u) \leq t$ in H , since $(c, c) \in [0, b] \times [0, c]$. This contradicts $C_{\max}^G(c, u) = C_{\max}^H(c, c, u)$.

So $a \geq c$ and we obtain

$$\begin{aligned} Q_1(A) &= Q_1([0, a]) \geq Q_1([0, c]) \\ &\geq Q_1([0, b])Q_1([0, c]) \\ &= Q_1 \otimes Q_1([0, b] \times [0, c]) = Q_1 \otimes Q_1(B). \quad \square \end{aligned}$$

Theorem 3 can be used to bound the distribution function of the makespan of a network from above and below. To this end, one must identify networks G_1, G_2 that “sandwich” the given network G in the sense that G_1 is a chain-minor of G and G is a chain-minor of G_2 . Then the unknown makespan distribution function F_G of G is “sandwiched” by those of G_1 and G_2 , i.e., $F_{G_1} \geq F_G \geq F_{G_2}$.

This brings up the question to identify networks G_1 and G_2 for which the distribution functions F_{G_1} and F_{G_2} are easier to evaluate. A proper choice is to consider series-parallel networks, since then the computation reduces to a sequence of convolutions and products of distribution functions. This is also the choice in the already quoted algorithms by Dodin, Kleindorfer, and Spelde.

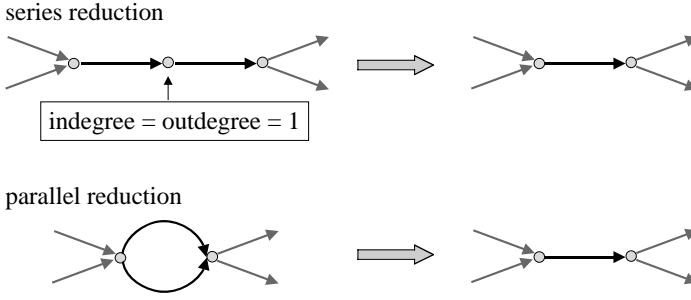


Fig. 8. Series and parallel reduction in arc diagrams.

4 Bounding with Series-Parallel Networks

A project network is *series-parallel*, if it can be reduced by a finite sequence of *series reductions* and *parallel reductions* to a network with only one job. In a series reduction, two jobs i, j that are “in series” in the arc diagram, i.e., the head of i is the tail of j and has indegree 1 and outdegree 1, are contracted to one job. In a parallel reduction, two jobs i, j that are “in parallel” in the arc diagram, i.e., have the same head and tail, are contracted to one job. These reductions are illustrated in Figure 8.

Series-parallel networks are a well-studied class of networks, see e.g. [24,15]. They can be recognized in linear time, and also a reduction sequence can be constructed in linear time, see [25].

For deterministic processing times, the makespan of a series-parallel network can be calculated along a reduction sequence by taking the sum of the processing times for a series reduction and the maximum for a parallel reduction, see Figure 9.

Since the processing times are stochastically independent, this favorable behavior generalizes to the stochastic case. One takes the distribution of the sum and the maximum of the processing times of the two jobs involved in a series and parallel reduction, respectively. In terms of distribution functions, these operations correspond to the *convolution* and the *pointwise product* of the distribution functions of the two jobs involved, see Figure 10.

It follows that the distribution function F_G of the makespan of a series-parallel network G can be calculated by convolutions and products “along” the reduction sequence of G . This approach belongs to the folklore in the field, see e.g. [13]. Although it suggests an easy and obvious algorithm, Möhring and Müller [17] show that the 2-state version of DF is still \mathcal{NP} -complete, but only in the weak sense. However, MEAN can in this case be solved in time polynomial in the largest number of values of the makespan of a network encountered in any series-parallel reduction sequence. This number may of course be exponential in the number of jobs.

Let DF-SP denote the restriction of DF to the class of series-parallel networks.

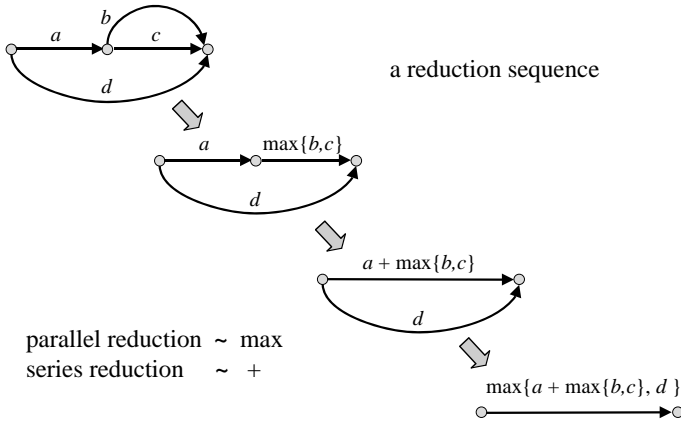


Fig. 9. Computing the makespan of a series-parallel network with deterministic processing times along a reduction sequence.

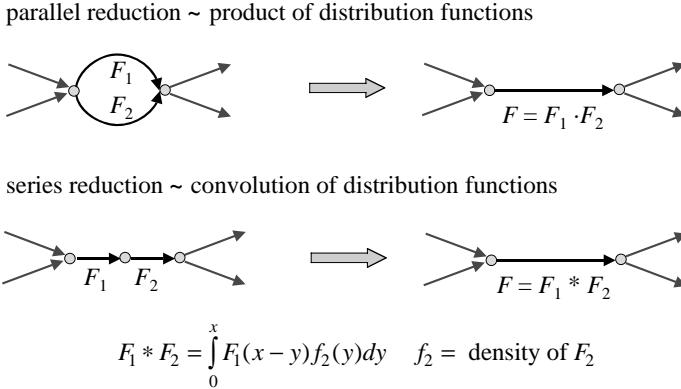


Fig. 10. Stochastic operations corresponding to series and parallel reductions.

Theorem 4. [17]

1. The two-state version of DF-SP is \mathcal{NP} -hard in the weak sense.
2. DF-SP with 0/1 processing times can be solved in $O(n^3)$ time.
3. DF-SP with arbitrary discrete processing time distributions can be solved in $O(N^2n)$ time, where N is the maximum number of distinct values that C_{\max} attains along the reduction sequence.

Proof. Statement 1 is proved by establishing a Turing reduction from PARTITION, which is known to be \mathcal{NP} -complete in the weak sense, to DF-SP. PARTITION is defined as

PARTITION: Given natural numbers a_1, \dots, a_n with $\sum a_i = 2b$, is there a partition I, J of $\{1, \dots, n\}$ with $\sum_{i \in I} a_i = \sum_{i \in J} a_i = b$?

Now, given an instance of PARTITION as above, we construct a network G consisting of only one path with n jobs $1, \dots, n$. Job j may attain the processing times 0 and a_j , each with probability $1/2$. Obviously, this is a polynomial transformation and G is series-parallel.

Now suppose that there exists a polynomial algorithm A for calculating the value $F_G(t)$ for arbitrary t . Then apply it to the values $t_1 := b$ and $t_2 := b - 1$. If there is a subset $I \subseteq \{1, \dots, n\}$ of the PARTITION instance with $\sum_{i \in I} a_i = b$, then the project duration C_{\max} attains the value b with positive probability $Q(C_{\max} = b)$ (the realization p with $p_i = a_i$ if $i \in I$ and $p_i = 0$ otherwise has $C_{\max}(p) = b$ and probability $\frac{1}{2^n}$). Hence F_G will have a jump at $t = b$ and the values $F_G(t_1)$ and $F_G(t_2)$ differ. If there is no such subset I , there is no jump, and $F_G(t_1) = F_G(t_2)$. Hence PARTITION is Turing reducible to DP-SP.

To show statement 3., we will assume that every discrete distribution function F is represented by a sorted list of their jumps $t_1 < t_2 < \dots < t_\ell$ with the associated values $F(t_1), \dots, F(t_\ell)$. Then a value $F(t)$ can be computed by a simple scan through the list in $O(\ell)$ time.

Now observe that, by assumption, every distribution function, the given ones and those calculated along the reduction sequence have at most N jumps. Hence calculating the product $F_i \cdot F_j$ of two distribution functions takes $O(N)$ time, while calculating the convolution $F_i * F_j$ takes $O(N^2)$ time. Since there are $n - 1$ reductions in the reduction sequence, F_G can be calculated in $O(N^2 \cdot n)$ time.

If all processing times are 0 or 1, we have $N \leq n + 1$ and thus $O(N^2 \cdot n) = O(n^3)$. This shows 2. \square

Theorem 4 shows that even for series-parallel networks G , evaluating the makespan distribution function F_G is as least as difficult as PARTITION.

This is essentially due to the fact that the iterated convolutions and products along a reduction sequence may result in a discrete processing time distribution with a number of distinct values N that is exponential in the input size. However, even then it is possible to solve DF in pseudo-polynomial time, which is not possible for arbitrary partial orders, since DF is $\#\mathcal{P}$ -complete in general.

We can conclude that series-parallel orders are easier to handle but still have a certain inherent complexity that—within the reduction approach—depends on the distributions generated on the way.

5 Specific Bounds

The very general bounding principle from the previous section covers the mentioned specific bounds of Kleindorfer, Spelde, Dodin and others. We will demonstrate this in two cases.

5.1 The Bound of Dodin

The algorithm of Dodin [5] works on the arc diagram D of the project network. A rough sketch is given in Figure 11 below. It carries out series and/or parallel

Input: Stochastic project network D as arc diagram

Output: A lower bound on the makespan distribution function of D

while D has more than one arc **do**

if a series reduction is possible **then** apply one

else if a parallel reduction is possible **then** apply one

else find

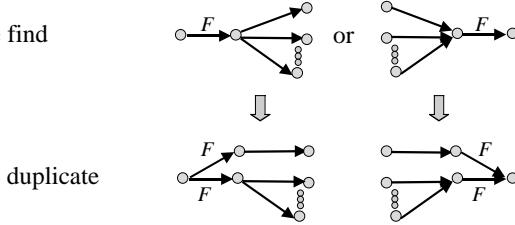


Fig. 11. The algorithm of Dodin.

reductions as long as possible. When this process stops and there are still more than one arc left, the algorithm performs an elementary duplication as depicted in Figure 11. That is, it identifies a node v of D with exactly one incoming arc i and more than one outgoing arc j_1, \dots, j_k (or, alternatively, a node with the dual situation). The incoming arc i is then used for an elementary duplication by making j_1 the only successor of the first copy of i and j_2, \dots, j_k the successors of the second copy of i (symmetrically in the alternative dual situation). The copies of i get the same distribution function as i .

An example is given in Figure 12. If one maintains an arithmetic expression with the graph operations ($+$ for a series reduction, \cdot for a parallel reduction, and the identity for a duplication), then the resulting distribution function F can be read off directly from the final string $((1 + 4) + (6 + 8) \cdot 7) \cdot ((2 \cdot (1 + 3) + 5) + 8)$ as $F = ((F_1 * F_4) * (F_6 * F_8) \cdot F_7) \cdot ((F_2 \cdot (F_1 * F_3) * F_5) * F_8)$.

Theorem 5. *Given a project network $D = (N, A)$ with random processing times as arc diagram, Dodin's algorithm computes a lower bound for the makespan distribution function of D , which is exact if D is series-parallel [5].*

The algorithm can be implemented to run in $\mathcal{O}(|A| \cdot (\text{conv}() + \text{prod}()) + |A| \cdot d_{\text{out}})$ time, where d_{out} denotes the maximum indegree (outdegree) of the nodes of D , and $\text{conv}()$ and $\text{prod}()$ denote the time for the convolution and product of two distribution functions, respectively [12].

Proof. First observe that the cases in the algorithm are well defined, i.e., if no series or parallel reduction is possible, then an elementary duplication can be carried out. To see this, consider a topological sort $1, 2, \dots$ of the nodes of D . Then node 2 has indegree 1 and outdegree > 1 , so the arc from 1 to 2 qualifies for duplication.

Series and parallel reductions do not change the makespan distribution. So Dodin's algorithm is exact if only these operations occur, i.e., if the network is series-parallel. After each duplication step, the current network contains the

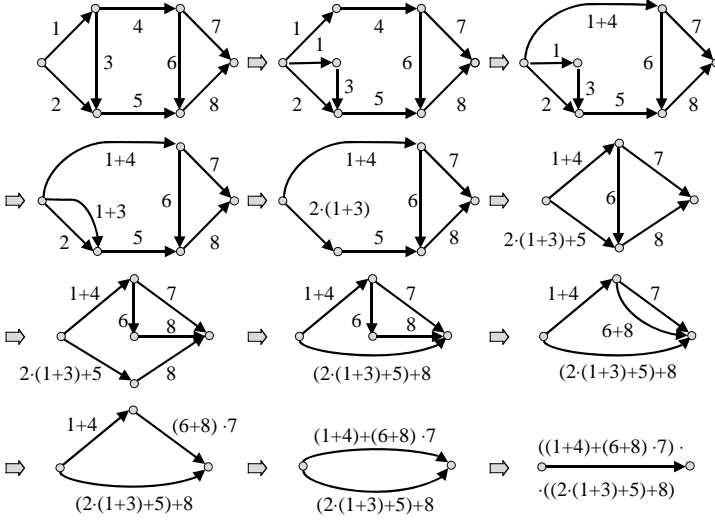


Fig. 12. An example of Dodin's algorithm. Series and parallel reductions are associated with the operations $+$ and \cdot , respectively.

previous one as a chain-minor. Hence the bounding property follows directly from applying Theorem 3 in every duplication step.

To analyze the runtime of the algorithm, we define a potential function $\Phi(D) = \sum_{v \in N} d_{out}(v) \cdot h(v) + \sum_{v \in N} d_{in}(v) \cdot \bar{h}(v)$. Here $d_{in}(v)$ and $d_{out}(v)$ denote the indegree and the outdegree of node v , while $h(v)$ and $\bar{h}(v)$ are the *height* of v (the least number of arcs on a directed path from the source of D to v) and the *dual height* of v (the least number of arcs on a directed path from v to the sink of D), respectively.

Obviously, $\Phi(D)$ decreases in two successive iteration by at least 1 since every duplication is followed by a series reduction. This shows that, even for arbitrary choices of reductions and duplications, the algorithm terminates after at most $\mathcal{O}(|N|^2) = \mathcal{O}(|A|^2) = \mathcal{O}(n^2)$ iterations. Theorem 3 then yields that the distribution function F of the last arc is a lower bound for the makespan distribution function F_D . When D is series-parallel, no duplication is made, and $F = F_D$.

The better runtime of $\mathcal{O}(|A| \cdot (\text{conv}() + \text{prod}())) + |A| \cdot d_{out}()$ results from a special choice of the arcs to duplicate and an “on the fly” handling of series and parallel reductions that are made locally possible by a duplication. In the first phase, all possible series and parallel reductions are performed in linear time by the series-parallel recognition algorithm of Valdes, Tarjan, and Lawler [25].

The second phase takes care of all duplications plus possible series and parallel reductions resulting from a duplication. To this end, one maintains a list L of nodes at which a series reduction becomes possible, which is initially empty. Every duplication is done with job j that is the arc from the source s to the first node v in a topological sort. The duplication is followed by the series re-

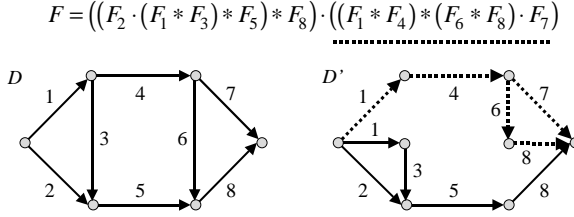


Fig. 13. Interpreting Dodin's bound as a chain-minor bound. The dotted part of D' corresponds to the dotted part of F .

duction involving one of the copies of j and, if possible, a parallel reduction involving the arc resulting from this series reduction. Checking for this parallel reduction takes $\mathcal{O}(d_{out})$ time. Furthermore one updates the list L since a duplication plus the added reductions may create new possibilities for series reductions, and empties it before the next duplication by carrying out the corresponding series reductions. As a result, all series and parallel reductions occurring in the second phase are either handled in $\mathcal{O}(d_{out})$ time together with a duplication, or by emptying list L which takes $\mathcal{O}(|A|)$ steps altogether. Using the modified potential function $\Phi'(D) = \sum_{v \in N \setminus \{source\}} d_{out}(v)$, one easily sees that the second phase has $\mathcal{O}(|A|)$ duplication steps. So altogether, the runtime is $\mathcal{O}(|A| \cdot (conv() + prod()) + |A| \cdot d_{out})$. \square

If one reverses the sequence of networks constructed by Dodin's algorithm by undoing all series and parallel reductions, but keeping all duplications, then the resulting network D' contains the given network D as a chain-minor, and the distribution function F computed by Dodin's algorithm equals the distribution function of the makespan of D' , i.e., $F = F_{D'}$. This is illustrated in Figure 13 for the example from Figure 12. This provides another proof of the bounding property of Dodin's algorithm.

This shows that Dodin's algorithm implicitly constructs a series-parallel network D' whose makespan distribution is taken as bound for F . In structural respect, the quality of this bound depends on how close D' is to the original network D . Bein, Kambarowski, and Stallmann [3] have introduced two measures that indicate the proximity of a network D to a series-parallel network D' . They are illustrated in Figure 14 below.

One of these measures, the number of *node reductions* is directly related to the process of elementary duplications. A reduction of a node v with $deg_{in}(v) = 1$ is defined as a sequence of $d_{out}(v) - 1$ elementary duplications of the arc ending in v as in Dodin's algorithm. (An analogous reduction can be performed if $d_{out}(v) = 1$). Bein, Kambarowski, and Stallmann [3] have shown that the least number $nr(D)$ of *node reductions* to obtain a series-parallel network from D can be computed in polynomial time by solving a vertex cover problem in an auxiliary, transitively orientable graph. So it provides an efficiently computable measure for the proximity to being series-parallel.

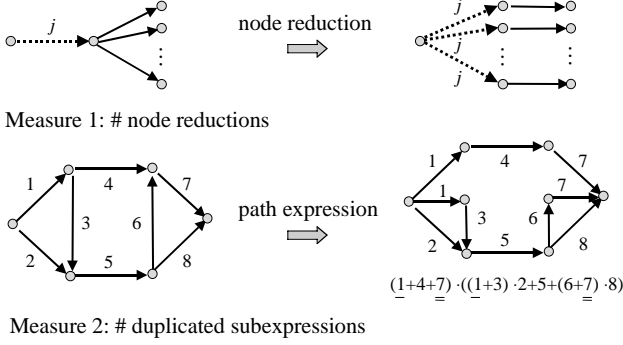


Fig. 14. Measuring the proximity to series-parallel networks.

The second measure they introduce, is the number of *duplicated subexpressions* in a “path expression” containing all the paths of D . In our terminology, the path expression is the arithmetic expression illustrated in Figures 12 and 13. Such an expression uniquely defines a series-parallel network, and, because of the required containment, the second measure is equivalent to the least number $ds(D)$ of duplicated subexpressions in the arithmetic expression of a series-parallel network containing D as a chain-minor. Kamburowski, and Stallmann [3] proved that $nr(D) \leq ds(D)$. Naumann [19] then showed that both measures are in fact equal, i.e., $nr(D) = ds(D)$ for every arc diagram D .

Of course, it is not only this distance that determines the quality of the bound, but also the actual processing time distributions.

5.2 The Bounds of Spelde

Spelde [23] presented both an upper and a lower bound that can be seen as direct applications of the chain-minor principle. The series-parallel networks D_1 and D_2 defining these bounds for D are very simple, they consist only of parallel chains as illustrated in Figure 15. D_1 defines the upper bound and consists of any system of pairwise disjoint chains of D covering all arcs of D , while D_2 , which defines the lower bound, uses all maximal chains of D (the extreme case in Figure 4).

Although these bounds seem very coarse at first glance, they have an important advantage. This lies in the fact that most other methods (including the bound by Dodin) assume that the processing time distributions are known. This is usually not the case in practice and often inhibits the use of these methods.

Spelde’s bounds offer a possibility to get around this information deficit. Since the bounding networks D_ℓ , $\ell = 1, 2$, are a parallel composition of paths, their makespan distribution functions F_{D_ℓ} are the product of the distribution functions of the lengths of these paths, say $F_{D_\ell} = F_1 \cdot F_2 \cdot \dots \cdot F_{N_\ell}$, where F_i is the distribution function of the i -th path.

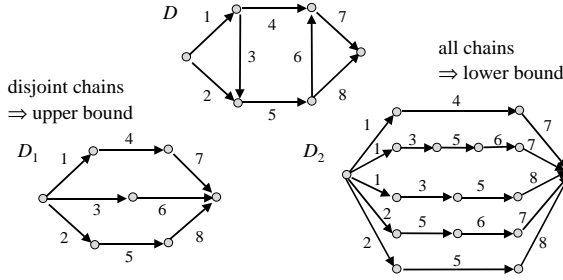


Fig. 15. The bounds of Spelde.

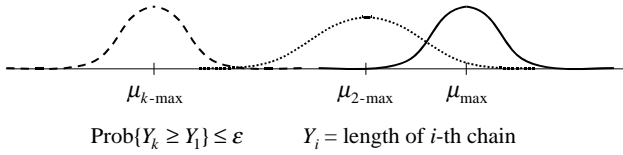


Fig. 16. The k longest paths in expectation suffice.

If the network D is large enough, i.e., all paths contain “enough” jobs, then, by the central limit theorem, every F_i is approximately a normal distribution function, whose mean μ_i and variance σ_i^2 are obtained as the sum of the means and variances of the processing times \mathbf{p}_j of all jobs j contained in the i -th path. This has been studied by many authors, see, e.g., [7,2,21]. Anklesaria and Drezner [2] as well as Sculli and Shum [21] considered multi-variant normal distributions in order to model correlations among paths. They also report on computational experiences on some small networks and obtained very promising results. In practice, we have observed that 10 jobs on a path already yield an excellent approximation of the path length distribution.

Hence it suffices to know the expected processing time $E(\mathbf{p}_j)$ and the variance $V(\mathbf{p}_j)$ of every job in order to calculate Spelde’s bounds. There is, however, a complication for D_2 since the number N_2 of all paths of D_2 may be exponential in the size of the given network D . This can be overcome by calculating the first k longest paths w.r.t. expected processing times $E(\mathbf{p}_j)$, until $\text{Prob}\{k\text{-th path is longer than 1st path}\} \leq \varepsilon$ for a given accuracy parameter ε (say $\varepsilon = 0.05$). If F_1, F_2, \dots, F_k are the normal distribution functions of these paths, then $F_{G_2} \approx F_1 \cdot F_2 \cdot \dots \cdot F_k$, see Figure 16.

In fact, this method contains the traditional PERT as a special case, since PERT only analyzes the distribution of the path with the longest expected path length.

Due to the reduced number of paths, one cannot guarantee the bounding property anymore. However, for a sufficiently large number of paths it is very likely that one still obtains a stochastic upper bound, at least for large quantiles.

In order to find the k longest paths, a variant of the algorithm proposed by Dodin [4] can be used.

5.3 Computational Experience

Ludwig, Möhring, and Stork [12] have implemented several bounds based on the chain-minor principle (Kleindorfer [10], Dodin [5], and Spelde [23]), and have made an extensive computational study of their bounding behavior on benchmark networks with up to 1200 jobs and a variety of different distributions.

In the code, the representation of the processing time distributions was a crucial task, as it had to be chosen with respect to an efficient and numerically stable computation of the sum and the maximum of random variables. Since the maximum operation is performed easily on distribution functions (it corresponds to the product), it was decided to represent each random variable by a piecewise linear approximation of its distribution function. A good approximation was obtained by choosing the supporting points arranged equidistantly on the ordinate. Thus, every distribution function is represented by its inverse. Furthermore, all distribution functions were represented by the same number of supporting points. This allows a simple implementation of the product and the convolution procedure. The product operation can be performed in linear time in the number sp of supporting points whereas the convolution requires quadratic running time in sp . In addition, these procedures turn out to compute reasonably stable results when a suitable number of supporting points is chosen.

In order to measure the quality of the computed bounds, they were compared with approximate distribution functions that were computed by extensive simulation with more than 1.5 million realizations for each instance.

Concerning runtime, the convolution operator consumed most of the computation time and therefore network transformations (which cause larger theoretical running times) are not the bottleneck. Spelde's lower bound based on k longest paths does not need any convolutions at all, and requires negligible computation times compared with the others (a factor of 20 to 50 times faster).

The quality of the computed bounds depended on the two test sets used. For one of them, the average relative error of the computed bounds is less than 1% for most of the considered instances, number of supporting points and distributions. For the second test set, the relative errors are a little larger, but still vary only between 1.6% and 3.7%. The maximum error occasionally exceeds 15%. However, this only occurs for the quantiles which are directly affected by the computation of boundary points at the computation of convolutions, i.e., the 0.01 and the 0.99 quantile. When disregarding these extreme quantiles, the maximum relative error reduces to 11%. In the case of Spelde's bound, $k \leq n/3$ paths were sufficient for this accuracy.

Perhaps the most important conclusion from this study is that Spelde's heuristic procedure based on the Central Limit Theorem provides excellent estimates at virtually no computational expense, and with only partial information about the processing time distributions. The approach is therefore a remarkable

alternative to (computationally costly) simulation techniques which are mostly used in practice.

References

1. V. Adlakha and V. Kulkarni. A classified bibliography of research on stochastic PERT networks: 1966-1987. *INFOR*, 27(3):272-296, 1989.
2. K. P. Anklesaria and Z. Drezner. A multivariate approach to estimating the completion time for PERT networks. *J. Opl. Res. Soc.*, 40:811-815, 1986.
3. W. W. Bein, J. Kamburowski, and M. F. M. Stallmann. Optimal reduction of two-terminal directed acyclic graphs. *SIAM J. Comput.*, 21:1112-1129, 1992.
4. B. Dodin. Determining the k most critical paths in PERT networks. *Oper. Res.*, 32(859-877), 1984.
5. B. Dodin. Bounding the project completion time distribution in PERT networks. *Oper. Res.*, 33:862-881, 1985.
6. D. R. Fulkerson. Expected critical path lengths in PERT networks. *Oper. Res.*, 10:808-817, 1962.
7. D. I. Golenko. *Statistische Methoden der Netzplantechnik*. B. G. Teubner, Stuttgart, 1972.
8. J. N. Hagstrom. Computational complexity of PERT problems. *Networks*, 18:139-147, 1988.
9. U. Heller. On the shortest overall duration in stochastic project networks. *Methods Oper. Res.*, 42:85-104, 1981.
10. G. B. Kleindorfer. Bounding distributions for a stochastic acyclic network. *Oper. Res.*, 19:1586-1601, 1971.
11. M. S. Krishnamoorthy and N. Deo. Complexity of the minimum-dummy-activities problem in a PERT network. *Networks*, 9:189-194, 1979.
12. A. Ludwig, R. H. Möhring, and F. Stork. A computational study on bounding the makespan distribution in stochastic project networks. Technical Report 609, Technische Universität Berlin, Fachbereich Mathematik, Berlin, Germany, 1998. To appear in and Annals of Oper. Res.
13. J. J. Martin. Distribution of the time through a directed acyclic network. *Oper. Res.*, 13:46-66, 1965.
14. I. Meilijson and A. Nadas. Convex majorization with an application to the length of critical paths. *J. Appl. Prob.*, 16:671-677, 1979.
15. R. H. Möhring. Computationally tractable classes of ordered sets. In I. Rival, editor, *Algorithms and Order*, Nato Advanced Study Institutes Series, pages 105-193. D. Reidel Publishing Company, Dordrecht, 1989.
16. R. H. Möhring. Scheduling under uncertainty: Optimizing against a randomizing adversary. In K. Jansen and S. Khuller, editors, *Approximation Algorithms for Combinatorial Optimization, Proceedings of the Third International Workshop APPROX 2000, Saarbrücken*, pages 15-26. Springer-Verlag, Lecture Notes in Computer Science, vol. 1913, 2000.
17. R. H. Möhring and R. Müller. A combinatorial approach to bound the distribution function of the makespan in stochastic project networks. Technical Report 610, Technische Universität Berlin, Fachbereich Mathematik, Berlin, Germany, 1998.
18. R. H. Möhring and F. J. Radermacher. The order-theoretic approach to scheduling: The stochastic case. In R. Słowiński and J. Węglarz, editors, *Advances in Project Scheduling*, pages 497-531. Elsevier Science B. V., Amsterdam, 1989.

19. V. Naumann. Measuring the distance to series-parallelity by path expressions. In E. W. Mayr, G. Schmidt, and G. Tinhofer, editors, *Proceedings 20th International Workshop on Graph-Theoretic Concepts in Computer Science WG'94*, pages 269–281. Springer-Verlag, Lecture Notes in Computer Science, vol. 903, 1995.
20. J. S. Provan and M. O. Ball. The complexity of counting cuts and of the probability that a graph is connected. *SIAM J. Comput.*, 12:777–788, 1983.
21. D. Sculli and Y. W. Shum. An approximate solution to the PERT problem. *Computers and Mathematics with Applications*, 21:1–7, 1991.
22. A. W. Shogan. Bounding distributions for a stochastic PERT network. *Networks*, 7:359–381, 1977.
23. H. G. Spelde. *Stochastische Netzpläne und ihre Anwendung im Baubetrieb*. PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen, 1976.
24. K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *J. Assoc. Comp. Mach.*, 29:623–641, 1982.
25. J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series-parallel digraphs. *SIAM J. Comput.*, 11:298–314, 1982.

Random Graphs, Random Triangle-Free Graphs, and Random Partial Orders

Hans Jürgen Prömel and Anusch Taraz

Institut für Informatik, Humboldt–Universität zu Berlin, D-10099 Berlin
`{proemel,taraz}@informatik.hu-berlin.de`

1 Introduction

While everybody seems to immediately understand and accept the commonly used model of a random graph – simply toss a coin for every edge to decide whether it is there – the situation gets harder when we require that the random graph must satisfy some additional constraints such as having no triangles or being transitive.

Here is the outline of this lecture. After this introduction and a few words about notation, Chapter 2 starts with a quick glance at the by now classical theory of random graphs. Among the many sights worth seeing there, we shall focus on results concerning the clique number and the chromatic number. One of the recurrent themes of this lecture is the question how these two parameters are related, in other words how tight the following trivial inequality is:

$$\omega(G) \leq \chi(G). \quad (1)$$

Obviously there are examples of graphs where ω and χ are not the same (there wouldn't be a theory of perfect graphs otherwise). More generally, it is of course well known that one can construct infinite families of graphs which have clique number 2 and arbitrarily high chromatic number; and furthermore, Erdős even proved the existence of graphs with arbitrarily high girth and arbitrarily high chromatic number. However, we will be concerned with the *typical* values of ω and χ : what is the clique number of a random graph, and what is its chromatic number? We will see that these questions have quite clear answers, and that the values of the two parameters lie far apart from each other.

Then, in Chapter 3, entering the area of random structures that have to satisfy side constraints, we look at what happens if we nail down the parameters: what is the difference between a random graph satisfying $\omega = 2$ and a random graph satisfying $\chi = 2$? Basically none, as we shall see: the number of those satisfying one but not the other requirement is negligible.

We then take a little digression and adopt an evolutionary point of view. So far we have only considered uniform probability distributions – which favour dense graphs, simply because there are so many of them. Now we take the edge density as a control or time parameter, and study the structure of random graphs as the density increases. An example: the chromatic number jumps from 2 to 3 when every edge has roughly probability $1/n$ (the first triangles appear at

roughly the same time). But it is not known when it jumps from 3 to 4, nor why it does so. And again, fixing one of the parameters: if we forbid triangles all the way through, then what happens to the chromatic number?

In Chapter 4 we change the setting. A graph is just a set with a binary relation. Now we require the relation to be transitive - in other words: instead of forbidding them, we explicitly require many triangles to be there. Thus we move from graphs to partial orders, but keep asking similar questions as before.

We shall need the following notation. Denote by $[n] := \{1, \dots, n\}$. A *graph property* is a family of graphs which is closed under isomorphism. For a graph property \mathcal{A} we let \mathcal{A}_n be those graphs in \mathcal{A} with vertex set $[n]$. The logarithm to base 2 will be denoted by \log . Throughout the entire lecture we will be somewhat sloppy by disregarding rounding of real numbers.

The following elementary bounds will be helpful and are displayed here for easier reference.

$$\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{en}{k}\right)^k, \quad \text{thus} \quad \log \binom{n}{k} = k \log n - k \log k + O(k). \quad (2)$$

For $0 < c < 1$,

$$\binom{m}{cm} = 2^{H(c)m + o(m)}, \quad (3)$$

where $H(c) = -c \log_2 c - (1-c) \log_2 (1-c)$ is the entropy function. For all x ,

$$1 - x \leq e^{-x}. \quad (4)$$

2 Random Graphs

2.1 General Properties of a Random Graph

Start with the empty graph on the vertex set $[n]$. Then insert each edge *independently* with probability $\frac{1}{2}$. The outcome of this random process is what we call a random graph and denote by $G_{n,1/2}$. Obviously, for a fixed graph $G \in \mathcal{G}_n$ (where \mathcal{G}_n is the set of all labeled graphs with vertex set $[n]$), the chance that $G_{n,1/2} = G$ is the same for all graphs G , namely $2^{-\binom{n}{2}}$, which is the reason why $G_{n,1/2}$ is referred to as the *uniform model*: instead of generating $G_{n,1/2}$ edge by edge, we could have simply chosen a graph at random, with equal probabilities, from \mathcal{G}_n .

In this section, we shall consider a variety of graph-theoretic properties and our general aim will be to see that *almost surely* a random graph admits these properties - by which we mean that, for some property \mathcal{A} , the probability that $G_{n,1/2}$ has \mathcal{A} converges to one as n tends to infinity. Since we are really working in a finite probability space, the probability in question is nothing else than the ratio $|\mathcal{A}_n|/|\mathcal{G}_n|$. So the above statement will sometimes be phrased as *almost all graphs have \mathcal{A}* ; justifying (or at least motivating) why we say that a random graph exhibits those properties which are typical for a graph on n vertices. We start with a simple example.

Proposition 1. $G_{n,1/2}$ almost surely has a triangle.

Proof. The probability that three given vertices do not form a triangle is $7/8$. Partition the vertex set into $n/3$ disjoint sets of three vertices each. The probability that $G_{n,1/2}$ does not have a triangle is thus at most $(7/8)^{n/3}$, which tends to zero as n tends to infinity.

Obviously, the proof shows just as well that $G_{n,1/2}$ almost surely contains a K_ℓ as long as ℓ is constant, which implies that $G_{n,1/2}$ almost surely contains every graph H of constant size as a subgraph. We shall see in a minute that this remains true in terms of *induced* subgraphs. To this end, we consider a property that might seem very artificial at first sight. Denote by $\mathcal{P}(s, t)$ the set of all graphs $G = (V, E)$ such that for every set $U \subseteq V$ with $|U| = s$ and every set $W \subseteq V$ with $|W| = t$, there exists a vertex $x \notin U \cup W$ which is connected to all vertices in U but to none in W .

Proposition 2. For every fixed s and t , almost surely $G_{n,1/2}$ has property $\mathcal{P}(s, t)$.

Proof. Fix two sets U and W as above. The probability that a specific vertex x satisfies the requirements is 2^{-s-t} . Hence the probability that no vertex x satisfies them is $(1 - 2^{-s-t})^{n-s-t}$, as there are $n - s - t$ candidates for x . Finally the probability that this happens for at least one pair of sets U, W is at most $n^{s+t}(1 - 2^{-s-t})^{n-s-t}$, which tends to zero as long as s, t are fixed.

One of the reasons for considering property $\mathcal{P}(s, t)$ is that it easily implies many (more natural) graph properties, as we shall see now.

Proposition 3. Let k be a fixed integer. Almost surely, $G_{n,1/2}$

- a) has minimum degree at least k ,
- b) is k -connected,
- c) has diameter 2,
- d) contains every graph H with at most k vertices as an induced subgraph.

Proof. These statements can be immediately derived from Proposition 2. We demonstrate this for part d) only. We can assume that $G_{n,1/2}$ has property $\mathcal{P}(s, t)$ for all $0 \leq s, t \leq k$, and that we have already “found” the first $i - 1$ vertices of H in $G_{n,1/2}$. Let v be the i -th vertex in H . We are looking for a vertex x in $G_{n,1/2}$ that can play the role of v , so among the first $i - 1$ vertices denote by U those which must be neighbours of x and by W those which must not. Now property $\mathcal{P}(s, t)$ (where we let $s = |U|$ and $t = |W|$) guarantees that we find a vertex x in $G_{n,1/2}$ with just the right connections.

Remarks. Property $\mathcal{P}(s, t)$ or rather Proposition 2 can be used to show that every first-order sentence about graphs satisfies a 0-1-law: it is either almost always or almost never true. Moreover, moving to the infinite without worrying about the definitions for this time, one can show that with probability one, a random countable graph has $\mathcal{P}(s, t)$ for all finite s and t , from which the following somewhat surprising result can be deduced: There is a countably infinite graph R , called the *Rado graph*, such that, with probability one, a random countably infinite graph is isomorphic to R .

2.2 Clique Number and Chromatic Number of a Random Graph

Having considered some graph properties, we now turn to the two graph parameters that we emphasized in the introduction: the clique number and the chromatic number. We have already seen that $\omega(G_{n,1/2}) \geq \ell$ for any constant ℓ . But how large can ℓ be as a function of n ? We introduce the random variable X_ℓ which denotes the number of ℓ -cliques in $G_{n,1/2}$. Then

$$\mathbb{E}[X_\ell] = \binom{n}{\ell} 2^{-\binom{\ell}{2}} \stackrel{(2)}{=} 2^{\ell \log n - \ell^2/2 - o(\ell^2)} \quad (5)$$

What we learn from this is the following: if we set $\ell = (2 + \beta) \log n$ for some constant $-1 \leq \beta \leq 1$, then (5) stands as

$$\mathbb{E}[X_\ell] = 2^{(-\beta - \frac{\beta^2}{2}) \log^2 n + o(\log^2 n)}, \quad (6)$$

implying that for positive β the expected value will converge to zero while for negative β it will tend to infinity. This indicates that we expect $\omega(G_{n,1/2})$ to be asymptotically $2 \log n$.

We will now prove that in fact $\omega(G_{n,1/2})$ is much more concentrated than this: we will show that $\omega(G_{n,1/2})$ takes on *only two* values with nonzero probability – a remarkably strong concentration result. For this we need the following two well-known tools.

Lemma 1 (Markov's inequality and Chebyshev's inequality).

Let X be a non-negative random variable. Then for any $t > 0$,

$$\mathbb{P}[X \geq t] \leq \frac{\mathbb{E}[X]}{t}, \quad \text{in particular,} \quad \mathbb{P}[X \geq 1] \leq \mathbb{E}[X]. \quad (7)$$

Denote by $\mu := \mathbb{E}[X]$ and by $\text{Var}[X] := \mathbb{E}[(X - \mu)^2] = \mathbb{E}[X^2] - \mu^2$. Then for any $t > 0$

$$\mathbb{P}[|X - \mu| \geq t] \leq \frac{\text{Var}[X]}{t^2}, \quad \text{in particular,} \quad \mathbb{P}[X = 0] \leq \frac{\mathbb{E}[X^2]}{\mu^2} - 1. \quad (8)$$

□

Theorem 1. *There exists a function $\ell_0 = \ell_0(n) \sim 2 \log n$ such that almost surely $\ell_0 - 1 \leq \omega(G_{n,1/2}) \leq \ell_0$.*

Proof. The pattern of the proof is the standard way to prove a concentration result: First we apply (7) to show that $\omega \leq \ell_0$ (the so-called first moment method) and then we use (8) to show that $\omega \geq \ell_0 - 1$ (the so-called second moment method).

We start by noticing that for $\ell \sim 2 \log n$

$$\frac{\mathbb{E}[X_{\ell+1}]}{\mathbb{E}[X_\ell]} = \frac{n - \ell}{\ell + 1} 2^{-\ell} \leq n 2^{-(1+o(1))2 \log n} = n^{-1+o(1)}, \quad (9)$$

which means that every time we increase ℓ by one, $\mathbb{E}[X_\ell]$ decreases by a factor of (roughly) n . So now we choose ℓ_0 so that

$$\mathbb{E}[X_{\ell_0+1}] \leq n^{-1/2} < \mathbb{E}[X_{\ell_0}]. \quad (10)$$

Looking at (6) shows that therefore $\ell_0 \sim 2 \log n$, and hence (9) and (10) imply that

$$n^{1/2-o(1)} \leq \mathbb{E}[X_{\ell_0-1}]. \quad (11)$$

By the choice of ℓ_0 , we have that $\mathbb{P}[\omega(G_{n,1/2}) \geq \ell_0 + 1] \stackrel{(7)}{\leq} \mathbb{E}[X_{\ell_0+1}] \stackrel{(10)}{\leq} 1/\sqrt{n} \rightarrow 0$, so $\omega(G_{n,1/2})$ is almost never larger than ℓ_0 .

Can $\omega(G_{n,1/2})$ be smaller than $\ell_0 - 1$? We let $\mu := \mathbb{E}[X_{\ell_0-1}] = \binom{n}{\ell_0-1} 2^{-\binom{\ell_0-1}{2}}$. In order to apply (8), we write $X_{\ell_0-1} = \sum_S Y_S$, where S runs over all sets of $\ell_0 - 1$ vertices, and Y_S is one if $G_{n,1/2}[S]$ is a clique and zero otherwise. Now

$$\begin{aligned} \mathbb{E}[X_{\ell_0-1}^2] &= \sum_S \mathbb{E}[Y_S^2] + \sum_{S \neq T} \mathbb{E}[Y_S Y_T] \\ &= \sum_S \mathbb{E}[Y_S] + \sum_{k=0}^{\ell_0-2} \sum_{S, T: |S \cap T|=k} \mathbb{P}[G_{n,1/2}[S] = K_{\ell_0-1} \wedge G_{n,1/2}[T] = K_{\ell_0-1}], \end{aligned}$$

thus

$$\begin{aligned} \frac{\mathbb{E}[X_{\ell_0-1}^2]}{\mu^2} &= \frac{\mu}{\mu^2} + \frac{\sum_{k=0}^{\ell_0-2} \binom{n}{\ell_0-1} \binom{\ell_0-1}{k} \binom{n-(\ell_0-1)}{\ell_0-1-k} 2^{-2\binom{\ell_0-1}{2} + \binom{k}{2}}}{\binom{n}{\ell_0-1}^2 2^{-2\binom{\ell_0-1}{2}}} \\ &\leq \frac{1}{\mu} + 1 + \sum_{k=1}^{\ell_0-2} \binom{\ell_0-1}{k} \binom{n-(\ell_0-1)}{\ell_0-1-k} 2^{\binom{k}{2}} / \binom{n}{\ell_0-1}. \end{aligned}$$

By a straightforward (but lengthy) computation the last sum can be shown to tend to zero. So together with the fact (11) that $\mu \rightarrow \infty$ this yields that

$$\mathbb{P}[\omega(G_{n,1/2}) < \ell_0 - 1] = \mathbb{P}[X_{\ell_0-1} = 0] \stackrel{(8)}{\leq} \frac{\mathbb{E}[Y^2]}{\mu^2} - 1 \rightarrow 0,$$

so almost surely $G_{n,1/2}$ has a clique of size $\ell_0 - 1$.

Having nailed down the clique number of $G_{n,1/2}$, we now move on to the chromatic number. Observe that Theorem 1 immediately implies the following (almost sure) lower bound:

$$\chi(G_{n,1/2}) \geq \frac{n}{\alpha(G_{n,1/2})} = \frac{n}{\omega(\bar{G}_{n,1/2})} \sim \frac{n}{2 \log n}. \quad (12)$$

The question whether (12) is sharp, i.e. whether $(1 + o(1))n/(2 \log n)$ colours almost surely suffice to colour $G_{n,1/2}$ was open for almost two decades until it was solved by Bollobás [7] in 1988:

Theorem 2. *Almost surely,*

$$\chi(G_{n,1/2}) \sim \frac{n}{2 \log n}.$$

Proof. Recall that in the proof of Theorem 1, we showed that

$$\mathbb{P}[\omega(G_{n,1/2}) < \ell_0(n) - 1] \rightarrow 0.$$

Using a much stronger large deviation inequality, namely the (generalized) Jan-son inequality (see e.g. [21]), this can be extended to show that

$$\mathbb{P}[\omega(G_{n,1/2}) < \ell_0(n) - 4] < e^{-n^{2+o(1)}}, \quad (13)$$

so here the probability not only tends to zero but does so very quickly. We will not prove (13) here. (The original proof of (13) relied on a very subtle application of combinatorial martingales and a large deviation inequality known as Azuma's inequality.)

Now we set $m = n/\log^2 n$ and observe that for a set S of size m the random graphs $G_{n,1/2}[S]$ and $G_{m,1/2}$ are really the same thing. So if we let $\ell = \ell_0(m) - 4$, then $\ell \sim 2 \log m$ and

$$\mathbb{P}[\alpha(G_{n,1/2}[S]) < \ell] = \mathbb{P}[\omega(G_{n,1/2}[S]) < \ell] \stackrel{(13)}{<} e^{-m^{2+o(1)}}.$$

As there are at most $2^n = 2^{m^{1+o(1)}}$ such sets S , we know that the probability that $\alpha(G_{n,1/2}[S]) < \ell$ for *some* set S is bounded by $2^{m^{1+o(1)}} e^{-m^{2+o(1)}} = o(1)$; in other words almost surely $G_{n,1/2}$ has the property that *every* set S of size m contains ℓ independent vertices.

It remains to show that a graph which has this property can be coloured with $(1 + o(1))n/(2 \log n)$ colours. Here is the algorithm which does it: choose any set S (of size m), pull out ℓ independent vertices and give them the same colour. Continue to do so, each time with a new colour, until less than m vertices remain - they all receive distinct colours. This way we use at most

$$\frac{n}{\ell} + m \sim \frac{n}{2 \log m} + \frac{n}{\log^2 n} = (1 + o(1)) \frac{n}{2 \log n} + o\left(\frac{n}{\log n}\right)$$

colours, which proves the theorem.

3 Random Triangle-Free Graphs

3.1 Random Triangle-Free Graphs Are Bipartite

In the last section we saw that for almost all graphs $\omega(G)$ and $\chi(G)$ are quite far apart from each other, the first being close to $2 \log n$ and the second close to $n/(2 \log n)$. In this section we shall see that if we restrict ourselves to triangle-free graphs however, then almost surely the clique number and the chromatic number are equal.

Once more, we return to our starting point (1) and consider the cases where $\chi(G) = 2$ and $\omega(G) = 2$ respectively. Let $\text{Col}(2)$ be the set of all bipartite graphs and let $\text{Forb}(K_3)$ be the set of all triangle-free graphs. Clearly $\text{Col}_n(2) \subset \text{Forb}_n(K_3)$; and we have seen in Proposition 1 that both sets are negligible in size when compared to the overall universe \mathcal{G}_n . Now we zoom in: suppose that the set $\text{Forb}_n(K_3)$ is our universe. What is the probability that a graph, drawn uniformly at random from the set of all triangle-free graphs, is bipartite? When approaching this question, one quickly becomes aware that determining the structure of a random element from $\text{Forb}_n(K_3)$ is much harder than that of a random element from \mathcal{G}_n . The reason for this fundamental problem is that the $G_{n,1/2}$ model owes much of its great success to the fact that it can be generated by a series of *independent* events. Obviously this independence now no longer exists: the uniform distribution of all triangle-free graphs cannot be modeled by independently inserting edges, and therefore many of the methods mentioned in Section 2 remain ineffective here.

One of the strategies to overcome these difficulties was first developed by Kleitman and Rothschild for the proof of Theorem 8, which we shall encounter later. Together with Erdős, they used it to prove the following theorem [15].

Theorem 3. *Almost all triangle-free graphs are bipartite.* □

Here we only give a very brief sketch of the proof technique. Our aim is to show that $|\text{Forb}_n(K_3)| = (1 + o(1)) \cdot |\text{Col}_n(2)|$. The first step is to define a collection of sets $\mathcal{R}_n^1, \dots, \mathcal{R}_n^k$ so that

$$\text{Forb}_n(K_3) \subseteq \text{Col}_n(2) \cup \mathcal{R}_n^1 \cup \dots \cup \mathcal{R}_n^k.$$

In the next step we bound $|\mathcal{R}_n^i|$ in terms of $|\text{Forb}_{n-x}(K_3)|$ for some appropriate $x \in \mathbb{N}$ and all i . By induction on n , we know that $|\text{Forb}_{n-x}(K_3)|$ is roughly the same as $|\text{Col}_{n-x}(2)|$, and from here to $|\text{Col}_n(2)|$ is easy because the number of bipartite graphs (and thus the ratio $|\text{Col}_n(2)|/|\text{Col}_{n-x}(2)|$) can be approximated much more easily than the number of triangle-free graphs. So in the end we have a bound on the cardinality of each set \mathcal{R}_n^i in terms of the cardinality of $\text{Col}_n(2)$ and thus we can show that all the \mathcal{R}_n^i are negligible compared to $\text{Col}_n(2)$.

Theorem 3 was generalized in two ways. Denote by $\text{Forb}(H)$ the set of all graphs not containing a copy of H and by $\text{Col}(\ell)$ the set of all ℓ -colourable graphs. For any constant ℓ , Lamken and Rothschild [29] showed that almost all graphs in $\text{Forb}(C_{2\ell+1})$ are 2-colourable, and Kolaitis, Prömel and Rothschild [27] proved that almost all graphs in $\text{Forb}(K_{\ell+1})$ are ℓ -colourable. A further extension, reuniting the latter two results, was obtained by Prömel and Steger [37] who characterized the family of forbidden graphs H for which such a statement holds. For a graph H with $\chi(H) = \ell + 1 \geq 3$,

$$|\text{Forb}_n(H)| = (1 + o(1)) \cdot |\text{Col}_n(\ell)|$$

holds if and only if H contains an edge whose deletion from H causes $\chi(H)$ to decrease.

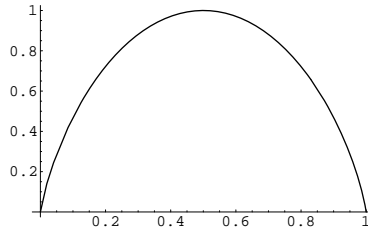


Fig. 1. $H(d)$

It is interesting that even the (negligible) class $\mathcal{R}_n = \mathcal{Forb}_n(K_3) \setminus \mathcal{Col}_n(2)$ which seems to avoid a structural result is in fact highly structured. Prömel, Schickinger and Steger [35,36] showed that almost every graph in \mathcal{R}_n can be made bipartite by removing only one vertex. Moreover, this can be iterated (for almost all of the remaining non-bipartite graphs it suffices to remove two vertices and so on) and generalized to ℓ -colourable graphs.

3.2 Evolution of Random Graphs

Let us step back a little bit and consider what we have done so far. One of our primary concerns was to analyze the properties we can almost surely expect from a graph which is chosen *uniformly at random* from the set of all graphs (or the set of all triangle-free graphs) and thus to describe the *global* structure that might not be common to all elements but certainly to the overwhelming majority.

The following observation might explain why this global structure may give a somewhat distorted picture. Recall that \mathcal{G}_n denotes the set of all graphs with vertex set $[n]$ and let $\mathcal{G}_{n,m}$ be those graphs in \mathcal{G}_n that have exactly m edges. Setting $m = d \cdot \binom{n}{2}$ for some constant $0 \leq d \leq 1$, we obtain that

$$|\mathcal{G}_{n,m}| = \binom{\binom{n}{2}}{m} \stackrel{(3)}{=} 2^{H(d)\binom{n}{2} + o(n^2)}.$$

Considering the entropy function $H(d)$ (see Figure 1), this immediately shows that almost all graphs have roughly $\frac{1}{2}\binom{n}{2}$ edges. This means that so far, really, we have only been analyzing the global structure of $\mathcal{G}_{n,m}$ for $m = (1 \pm \epsilon)n^2/4$ – which, admittedly, is by far the *largest* “slice” of \mathcal{G}_n but, after all, only *one*. Of course, in that slice the clique number may well typically be of order $2 \log n$ and the chromatic number of order $n/(2 \log n)$, but what if we focus on another slice?

We model this as follows. For some $0 < p < 1$ denote by $G_{n,p}$ the random graph obtained by inserting each edge *independently* with probability p . Notice that most of the times $p = p(n)$ will actually be a function of n . This brings us to the following question: *How does the structure of a typical graph gradually*

change as the probability distribution changes? This is the key question when investigating the evolution process of the random graph $G_{n,p}$. In a series of seminal papers Erdős and Rényi [16,17,18,19] investigated this process, thereby marking the starting point of the theory of random graphs. For many graph properties they noticed so-called phase transition phenomena: *Until p reaches a certain threshold, $G_{n,p}$ will almost surely not have the property in question, whereas beyond this threshold suddenly it will almost surely possess it.* Here we only briefly outline the main evolutionary phases as described by Erdős und Rényi in [17]. (More details and further references can be found in [6,21,22]).

Phase 1: $p \ll 1/n$. Here $G_{n,p}$ is a forest. Trees on k vertices appear for the first time when $p = \Theta(n^{-k/(k-1)})$.

Phase 2: $p \sim c/n$ where $0 < c < 1$. All connected components of $G_{n,p}$ are either trees or trees with one additional edge. The largest component is a tree of order $\Theta(\log n)$.

Phase 3: $p = 1/n$. Here the largest component, now called the *giant component*, already has order $n^{2/3}$. However by ‘slowing down’ the growth of p one sees that the evolution of $G_{n,p}$ is rapid but continuous.

Phase 4: $p \sim c/n$ where $c > 1$. This phase can roughly be described as the merging of the small components (most of which are trees) into the giant component, which now has $\Theta(n)$ vertices.

Phase 5: $p \gg \log n/n$. Here $G_{n,p}$ has just become connected, it contains a Hamilton cycle and all the vertices have asymptotically the same degree.

Now we will try to inspect a few of the above examples in more detail. Consider an arbitrary graph property \mathcal{A} . A function $t = t(n)$ is called the *threshold function* for \mathcal{A} , if, as n tends to infinity,

$$\begin{aligned} \frac{p(n)}{t(n)} \rightarrow 0 & \quad \Rightarrow \quad \mathbb{P}[G_{n,p} \in \mathcal{A}] \rightarrow 0, & \quad \text{and} \\ \frac{p(n)}{t(n)} \rightarrow \infty & \quad \Rightarrow \quad \mathbb{P}[G_{n,p} \in \mathcal{A}] \rightarrow 1. \end{aligned}$$

Strictly speaking, one ought to say *a* (rather than *the*) threshold function, since there may obviously be more than one. The first of the two above requirements is called the *0-statement*, while the second is referred to as the *1-statement*.

Theorem 4.

- a) The threshold function for the property of containing a triangle is $t(n) = 1/n$.
- b) The threshold function for the property of containing a cycle is $t(n) = 1/n$.
- c) For every fixed ℓ , the threshold function for the property of containing a subgraph which is isomorphic to K_ℓ is $t(n) = n^{-2/(\ell-1)}$.

Proof. We prove a) and b) both at the same time. The proof of c) will be omitted, since it is similar to that of a). The proof of a) and b) follows the same route as the proof of Theorem 1: we use the first moment method to prove the 0-statement and the second moment method for the 1-statement.

For $pn \rightarrow 0$, denote by X_ℓ the number of ℓ -cycles in $G_{n,p}$ and by X the total number of cycles in $G_{n,p}$. Obviously $\mathbb{E}[X_\ell] \leq n^\ell p^\ell$. We would like to show that

$\mathbb{P}[X \geq 1] < \epsilon$ for any $0 < \epsilon < 1$. For n large enough we have $p \leq \frac{\epsilon}{2n}$, so

$$\mathbb{P}[X_3 \geq 1] \leq \mathbb{P}[X \geq 1] \stackrel{(7)}{\leq} \mathbb{E}[X] = \sum_{\ell=3}^n \mathbb{E}[X_\ell] \leq \sum_{\ell=3}^n (\epsilon/2)^\ell < \epsilon,$$

which proves the 0-statement of a) and b).

As to the 1-statement where we are concerned with the case $pn \rightarrow \infty$, we denote by Y the number of triangles in $G_{n,p}$ and let $\mu = \mathbb{E}[Y] = \Theta(n^3 p^3)$. Similarly to the second part of the proof of Theorem 1, we write $Y = \sum_S Y_S$, where S runs over all sets of three vertices, and Y_S is one if $G_{n,p}[S] = K_3$ and zero otherwise. Again, we have that

$$\mathbb{E}[Y^2] = \sum_S \mathbb{E}[Y_S] + \sum_{k=0}^2 \sum_{S,T: |S \cap T|=k} \mathbb{P}[G_{n,p}[S] = K_3 \wedge G_{n,p}[T] = K_3],$$

hence

$$\frac{\mathbb{E}[Y^2]}{\mu^2} = \frac{\Theta(n^3 p^3 + n^6 p^6 + n^5 p^6 + n^4 p^5)}{\Theta(n^6 p^6)} = \Theta\left(\frac{1}{n^3 p^3} + 1 + \frac{1}{n} + \frac{1}{n^2 p}\right).$$

As $n^2 p \geq np \rightarrow \infty$, the last expression tends to one, and we can conclude that

$$\mathbb{P}[Y = 0] \stackrel{(8)}{\leq} \frac{\mathbb{E}[Y^2]}{\mu^2} - 1 \rightarrow 0,$$

so almost surely $G_{n,p}$ has a triangle (and therefore a cycle).

Remark. Bollobás and Thomason [10] proved that in fact every graph property which is closed with respect to taking subgraphs has a threshold.

Now let us see what Theorem 4 tells us about the evolution of random graphs in terms of the clique number and the chromatic number. For $p \ll 1/n^2$ the random graph $G_{n,p}$ will almost surely be empty. Moving on, Theorem 4 b) tells us that

$$\text{for } 1/n^2 \ll p \ll 1/n, \text{ almost surely } \chi(G_{n,p}) = \omega(G_{n,p}) = 2.$$

On the other hand, Theorem 4 a) implies that

$$\text{for } p \gg 1/n, \text{ almost surely } \chi(G_{n,p}) \geq \omega(G_{n,p}) \geq 3, \quad (14)$$

so at the point $p = 1/n$ both $\omega(G_{n,p})$ and $\chi(G_{n,p})$ jump from 2 to (at least) 3.

But what can we say about $\omega(G_{n,p})$ and $\chi(G_{n,p})$ when $p = c/n$ for some constant c ? Observe that due to our definition of a threshold function $t(n)$, Theorem 4 does not reveal any information for this case: we only know what $G_{n,p}$ looks like when $p(n)$ is *much* smaller or *much* larger than $t(n)$.

Theorem 5. For $p = c/n$, where $c > 0$ is fixed,

$$\mathbb{P}[\omega(G_{n,p}) > 2] \rightarrow 1 - e^{-c^3/6}.$$

□

Theorem 6. For $p = c/n$, where $0 < c < 1$ is fixed,

$$\mathbb{P}[\chi(G_{n,p}) > 2] \rightarrow 1 - e^{c/2}((1-c)/(1+c))^{1/4},$$

while for $p = 1/n$

$$\mathbb{P}[\chi(G_{n,p}) > 2] \rightarrow 1.$$

□

We state these two theorems without proof. Here we do not care so much about the formulae but would like to emphasize that they illustrate the different nature of the thresholds: the threshold in Theorem 5 is what we call *coarse*, which means that for any constant $c > 0$ the (limit of the) probability that $G_{n,ct(n)}$ has the required property lies strictly between 0 and 1. In contrast, the threshold in Theorem 6 is coarse on the left hand side and *sharp* on the right hand side: for any $\epsilon > 0$, the (limit of the) probability that $G_{n,(1+\epsilon)t(n)}$ has the required property is 1. Soon we will see that the property “ $\chi(G) > 3$ ” has a threshold which is sharp on both sides.

Coming back to the interval $p \gg 1/n$ and (14), can we say anything about $\omega(G_{n,p})$ and $\chi(G_{n,p})$ other than that they are at least three? Theorem 4 c) implies that

$$\text{for } 1/n \ll p \ll 1/n^{2/3}, \text{ almost surely } \omega(G_{n,p}) = 3,$$

so might this be true for $\chi(G_{n,p})$, too? On the other hand, we have seen that much later, namely at the point $p = 1/2$, $\omega(G_{n,p})$ is much smaller than $\chi(G_{n,p})$, so this indicates that the chromatic number increases faster than the clique number. Indeed, as we shall see now, computing the expected number of colourings shows that in fact for *every* constant k , the threshold for the property of (non-) k -colourability is given by $t(n) = 1/n$.

Proposition 4. For every constant k there exists a constant c_k such that if $p > c_k/n$ then almost surely $\chi(G_{n,p}) > k$.

Proof. For a k -partition $\pi : V_1 \cup \dots \cup V_k = V$ of the vertex set, the number of forbidden edges for π to be a k colouring is $\iota(\pi) := \sum_{i=1}^k \binom{|V_i|}{2}$ and it is easy to see that ι is minimal if all partition classes have the same size, i.e.

$$\iota(\pi) \geq k \frac{n}{2k} \left(\frac{n}{k} - 1 \right) > \frac{n^2}{3k}.$$

Now denote by X the number of k -colourings of $G_{n,p}$. Then, as $(1-p)^{\iota(\pi)}$ is the probability that π is a colouring of $G_{n,p}$,

$$\mathbb{E}[X] = \sum_{\pi} (1-p)^{\iota(\pi)} \stackrel{(4)}{\leq} k^n e^{-p \frac{n^2}{3k}} = k^n e^{-\frac{c_k}{n} \frac{n^2}{3k}} = \left(k e^{-\frac{c_k}{3k}} \right)^n,$$

which tends to zero if c_k is sufficiently large. So, by (7), the proof is complete.

Note that this seems to indicate (but doesn't imply) that, for $k > 2$, the property non- k -colourability has a sharp threshold. This was indeed proven by Achlioptas and Friedgut [1], using a recent result [3] on necessary and sufficient conditions for a property to have a sharp threshold. They were thus able to come closer to an old question of Erdős [16] who had conjectured that, for any $k > 2$, there exists a constant c_k so that the threshold for non- k -colourability is sharp and given by $t(n) = c_k n$.

Confirming the second part of this conjecture and actually determining c_k , even for $k = 3$, remains an open and challenging problem. Assuming that the c_k exist, the proof of Proposition 4 immediately gives, e.g., that $c_3 < 9.89$. By minor and not so minor modifications of the first moment method, this can be improved to $c_3 < 4.99$, the current record [28].

The question of lower bounds reveals an interesting story which comes in two parts. The first approach is based on the elementary fact that if a graph does not contain a subgraph of minimum degree k (a so-called k -core), then it must be k -colourable. Therefore the threshold $t'_k(n)$ for the appearance of a k -core is a lower-bound for the threshold of non- k -colourability. A major series of contributions led to the discovery [34] that $t'_k(n)$ is a sharp threshold and that, e.g., $t'_3(n) = 3.35.../n$. In order to show that this bound does not tell the truth about non- k -colourability – in other words: trying to show that $G_{n,p}$ remains k -colourable even after the emergence of the k -core – Achlioptas and Molloy [2] took a more constructive approach in that they analyzed the behaviour of a greedy colouring heuristic with input $G_{n,p}$. At the beginning, the algorithm assigns palettes with k colours to each vertex and then in each round colours one of those vertices with the highest risk of running out of colours. This procedure can be shown to successfully 3-color $G_{n,p}$ almost surely until $p = 3.84/n$. So the state of the art, for $k = 3$, is $3.84 < c_3 < 4.99$.

We conclude this section with the following result of Łuczak [30] which determines the behaviour of $\chi(G_{n,p})$ for $1/n \ll p \ll 1$.

Theorem 7. *Suppose $\epsilon > 0$ is fixed. Then for $p = c(n)/n$ with $c_\epsilon \leq c(n) = o(n)$ and a sufficiently large constant c_ϵ , almost surely*

$$\frac{c(n)}{2 \log c(n)} < \chi(G_{n,p}) < (1 + \epsilon) \frac{c(n)}{2 \log c(n)}.$$

□

3.3 Evolution of Random Triangle-Free Graphs

Now we turn our attention to what could be called *constrained* evolution. Thus instead of considering a graph chosen at random from the set of *all* graphs on n vertices, we consider only those which are triangle-free. We have already seen in Theorem 3 that almost all triangle-free graphs are bipartite. But is this also true if we focus our attention on an early evolutionary stage? Let us choose a random graph $T_{n,m}$ uniformly, not from $\mathcal{Forb}_n(K_3)$ but from $\mathcal{Forb}_{n,m}(K_3)$, the set of all triangle-free graphs with vertex set $[n]$ that have m edges.

Observe that for $m = o(n)$, we know by Theorem 4 b) that $T_{n,m}$ must be bipartite. Also, by Turán's theorem, the same is true for $m = n^2/4$. But what happens in between? In 1996, Prömel and Steger [38] proved that $T_{n,m}$ is almost *never* bipartite for $cn \leq m \leq c'n^{3/2}$, but almost *always* bipartite for $m \geq cn^{7/4} \log n$.

Interestingly, this already indicated that the evolution of random triangle-free graphs exhibits *two* phase transitions with respect to being bipartite: first it is almost surely bipartite, then it is not, and then it is once again.

Moving a step closer to closing the gap in the interval $n^{3/2} \ll m \ll n^{7/4} \log n$, Łuczak [31] proved that here $T_{n,m}$ is typically at least *almost-bipartite*, i.e. for every $\delta > 0$ there exists a constant c , such that for $m \geq cn^{3/2}$ the graph $T_{n,m}$ can be made bipartite by removing at most δm edges.

Based on this, Osthus, Prömel and Taraz [32] recently succeeded in completing the picture (see Figure 2). They showed that the second threshold is sharp (in contrast to the first) and given by

$$t(n) := \sqrt{3}/4 \, n^{3/2} \sqrt{\log n}.$$

(Independently, Steger [42] showed that the Kleitman–Rothschild method can also be used to prove that the threshold has order $n^{3/2} \sqrt{\log n}$.)

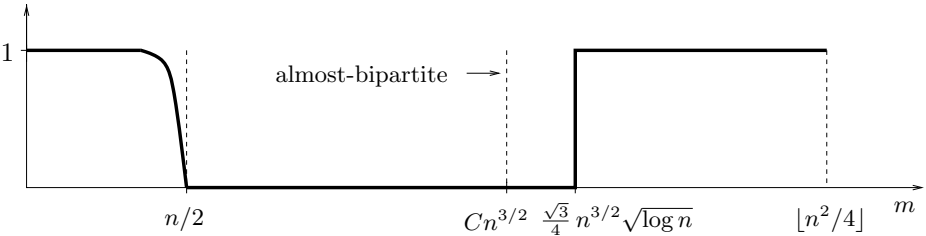


Fig. 2. The proportion of triangle-free graphs with n vertices and m edges which are bipartite, as $n \rightarrow \infty$

This leaves us with the following question: what is the chromatic number of $T_{n,m}$ when $cn \leq m \leq t(n)$? A first and simple approach shows that in the first part of the interval, where $cn \leq m \leq cn^{5/4}$, $T_{n,m}$ almost surely has no independent sets of size $2 \frac{n^2}{m} \log \frac{2m}{n}$. This implies a lower bound for $\chi(T_{n,m})$ which is only smaller than $\chi(G_{n,m})$ by a factor of 2.

3.4 Random Maximal Triangle-Free Graphs

Having looked at the uniform model in the previous section, in this short section we shall discuss a very natural dynamic model of generating random graphs which satisfy some sort of side-constraint such as, e.g., being triangle-free.

Consider the following random process. Select a graph H of constant size and choose a random ordering of the edges of the complete graph with vertex set $[n]$. Starting with no edges present, try to add the edges in the prescribed order, under the condition that no copy of H is created. Denote by G the final graph that has been constructed when all edges have tried to appear. Obviously G is a maximal H -free graph. What can we say about the structure of G ? In particular, how many edges does G have with high probability?

This question was raised by Bollobás and Erdős. Ruciński and Wormald [41] considered the case where H is a star with $d + 1$ leaves; in other words their process generated a graph with maximum degree at most d . They could show that almost surely the final graph G is almost surely *extremal* in the sense that all vertices have degree d with the possible exception of one vertex which has degree at least $d - 1$. Next, Erdős, Suen and Winkler [20] treated the case where H is a triangle. Here, the final graph almost surely has only $n^{3/2+o(1)}$ edges and is thus far from extremal. However, the seemingly more restrictive process, where *all* cycles of odd length are forbidden (i.e. the graph being generated is kept bipartite), almost surely leads to a final graph with $\Theta(n^2)$ edges. Intuitively spoken, this is due to the fact that as the colour classes of the graph grow, they tend to balance themselves, because a vertex which hasn't been assigned to a colour class yet is more likely to become connected first to the class which is currently larger, and thus ends up more often in the class which is currently smaller.

Recently, Osthus and Taraz [33] considered the above process for a more general family of graphs H . Denote by

$$m_2(H) = \max_{H' \subseteq H, |V(H')| > 2} \frac{|E(H')| - 1}{|V(H')| - 2}$$

the so-called 2-density of H . If H is strictly 2-balanced, i.e. its 2-density is strictly larger than the 2-density of any of its subgraphs (which is true for example when H is a cycle or a complete graph), then the final graph G almost surely has $n^{2-1/m_2(H)+o(1)}$ edges. (The cases where H is a K_4 or a C_4 were proven independently by Bollobás and Riordan [9].)

What about arbitrary graphs H ? Take the case where H is the disjoint union of a triangle and a quadrilateral. Now in our process, either a triangle or a quadrilateral will appear first with roughly equal probabilities. But if, say, the triangle is first, we will from now on forbid all quadrilaterals, and vice versa, and the outcomes will be decidedly different. This illustrates why one cannot expect a concentration result for every graph H .

We end this section by a tangential remark concerning a central result. Processes like the one above can be viewed as randomized algorithms that almost always generate objects with particular properties. When trying to prove the existence of certain objects, it suffices to ask for a merely positive probability, and, instead, push the required properties to the extreme. In 1995, Kim [23] used a similar (but more complicated) process to prove the existence of triangle-free graphs without large independent sets. He thereby obtained his celebrated lower

bound on the Ramsey number $R(3, t)$ which solved a problem that had been open for several decades.

4 Random Partial Orders

4.1 Random Partial Orders Have Height Three

When counting partial orders, we immediately notice the same obstacle as when counting graphs with forbidden subgraphs, namely the loss of independence: once we have decided that the comparable pairs $x < y$ and $y < z$ are to be present, we have no choice but to insert the comparable pair $x < z$ in order to guarantee transitivity.

Denote by \mathcal{P} the family of *all* labeled posets and by \mathcal{P}_n those posets from \mathcal{P} with point set $[n]$. Let us first look for an easy lower bound on $|\mathcal{P}_n|$. It will suffice to consider posets with height 2, so fix two antichains X and Y , each on $\frac{n}{2}$ points. In this case there are no problems with transitivity, so decide *independently* for each pair $(x, y) \in X \times Y$ whether or not $x < y$ should hold. Hence

$$|\mathcal{P}_n| \geq 2^{\frac{n^2}{4}}.$$

(Notice that here we didn't even care for the $\binom{n}{n/2} = O(2^n)$ ways in which we could have chosen the points which go to, say, X .) Kleitman and Rothschild [24,25] managed to show that this bound is already tight with respect to the leading term in the exponent. They showed that

$$|\mathcal{P}_n| = 2^{n^2/4 + \frac{3n}{2} + O(\log n)}. \quad (15)$$

The two papers leading to this result mark the beginning of the so-called Kleitman–Rothschild method that we already discussed in connection with Theorem 3. As we saw there, this method can be used to show that, for two sets $\mathcal{A} \subset \mathcal{B}$, almost every object in \mathcal{B} also belongs to \mathcal{A} . In our setting here, \mathcal{P} plays the role of \mathcal{B} . The class \mathcal{A} is defined as follows: the posets contained in \mathcal{A} all consist of three disjoint antichains X_1 , X_2 and X_3 , where both X_1 and X_3 have roughly $n/4$ points, all points in X_3 are above all points in X_1 , and whenever $x < y$ for two points $x \in X_i$ and $y \in X_j$ we must have $i < j$. Following the terminology used in [12], such a poset is called a *3-layer* poset. So on their way to (15) Kleitman and Rothschild proved the following remarkable theorem.

Theorem 8. *Almost all posets are 3-layer posets.* □

Let us immediately point out another consequence of this result. In a certain way, dimension has a similar meaning for partial orders as the chromatic number had for graphs. Hence the question of determining the dimension of a random partial order arises quite naturally. Now Theorem 8 implies that the dimension of a random partial order is that of a random 3-layer poset from the class \mathcal{A} above: let each comparable pair $x_1 < x_2$ or $x_2 < x_3$ (where $x_i \in X_i$) exist with probability $\frac{1}{2}$. This way, we recover the independence that has been lost through transitivity. In 1991, Erdős, Kierstead and Trotter [14] used this characterization to prove the following theorem.

Theorem 9. *There exist constants $c_1, c_2 > 0$ such that for almost every poset $P \in \mathcal{P}_n$*

$$\frac{n}{4}\left(1 - \frac{c_1}{\log n}\right) \leq \dim(P) \leq \frac{n}{4}\left(1 - \frac{c_2}{\log n}\right).$$

□

For the lower bound in this theorem it suffices to consider a random bipartite poset whose levels Y_1 and Y_2 both have $n/4$ points and where $y_1 < y_2$ with probability $1/2$ for any $y_1 \in Y_1$ and $y_2 \in Y_2$. Note that we can obtain such a random bipartite poset e.g. by choosing a random poset from \mathcal{A} and considering X_1 and half of X_2 . The upper bound is obtained by a probabilistic construction of a realizer whose linear extensions are chosen in such a way that their intersection almost surely guarantees any two points from the same layer to be incomparable.

We return to Theorem 8. It is in many ways something like the poset-counterpart to Theorem 3 and seems at first glance similarly ‘narrow minded’. As Brightwell [11] puts it:

[Theorem 8] has been widely viewed as a negative result. It says that almost every partial order has height three, and even more, is *ranked*, that is, all the maximal chains have the same length (namely, three). Somehow this does not conform to the practising mathematician’s view of what a “typical” partial order ought to look like.

This explains, at least in parts, the development of other models of random partial orders such as random bipartite orders, random graph orders or random k -dimensional orders (for an overview on these models see [11,43,44]). Here we merely sketch an introduction to random graph orders.

Suppose we were given a graph G together with a linear order $<$ on the vertex set of G . Consider first only those comparable pairs $i <_G j$ where i and j are vertices, $i < j$ and $\{i, j\}$ is an edge in G . The associated poset $<_G$ on the vertex set is then defined as the transitive closure of $<_G$. Then a *random graph order*, usually denoted by $P_{n,p}$, is the poset associated with $G_{n,p}$ and an arbitrary linear order in the above way.

From the earlier discussion of $G_{n,p}$, it is intuitively obvious, at least if p is a constant, that there is a big difference between a typical random poset $P_{n,p}$ and a typical random poset from the uniform model: the height of $P_{n,p}$ is much larger and its width is much smaller.

Random graph orders were introduced as such by Albert and Frieze [5] even though they had implicitly appeared earlier elsewhere. As an example of an interesting problem on random graph orders, we briefly address what seems to be an elementary question: For a given p , *how many comparable pairs does $P_{n,p}$ have?* We use this question to illustrate the fact that knowledge of the structure of random elements helps with the running time analysis of algorithms. Suppose one wanted to push the idea of “sorting in parallel” even further than the well-known AKS sorting-networks [4] which take $O(\log n)$ rounds, each with $O(n)$ comparisons. Namely, one attempts to make m comparisons in each round for some number $n \ll m \ll \binom{n}{2}$. Let’s say that the keys to be sorted are given by

$x_1 < \dots < x_n$. Choose p such that $p \cdot \binom{n}{2} = m$, pick a random graph $G_{n,p}$ on the keys and compare adjacent vertices. Now it is the underlying linear order that we want to recover from the information obtained via the comparisons *and* the transitive information that can be deduced from them. Hence the crucial point that determines the expected running time of this approach is: *the more comparable pairs we have in $<_G$ the faster the algorithm will be*. In this context, Bollobás and Brightwell [8] could show e.g. that for $p \geq \log^2 n/n$ the number of comparable pairs in $P_{n,p}$ is $\binom{n}{2} - o(n^2)$. Thus on average, one can sort almost all keys in a single round, using $O(n \log^2 n)$ comparisons.

4.2 Evolution of Random Partial Orders

Recall Theorem 8 which states that almost all posets are 3-layer posets. Compare this to Theorem 3: almost every triangle-free graph is bipartite. But, as we learned in Figure 1, this is “only” true because it is true for dense graphs. Now view the situation of partial orders from the same angle: what does a typical poset on n points *with a given number of comparable pairs* look like? And: *How many such posets are there?*

Let us introduce some notation. Denote by $\mathcal{P}_n(d)$ the family of all posets in \mathcal{P}_n with $d \cdot n^2$ comparable pairs, and let

$$c(d) := \lim_{n \rightarrow \infty} \frac{\log |\mathcal{P}_n(d)|}{n^2},$$

provided that the limit exists. (Strangely enough, the question of determining $c(d)$ seems to have first arisen in quite a different context: in 1978, Dhar [13] suggested that partial orders can represent the states of a certain model of lattice gas with energy proportional to the number of comparable pairs in the order.) Dhar [13] as well as Kleitman and Rothschild [26] determined $c(d)$ in the interval $0 < d \leq 3/16$ more than 20 years ago, but it was only recently that Prömel, Steger and Taraz [39,40] succeeded in describing what happens for $d > 3/16$. Let us explain the approach that has been taken here in order to compute $c(d)$. Recall from (15) that for any d we must have

$$c(d) \leq \frac{1}{4}. \tag{16}$$

For every $0 < d < \frac{1}{2}$ we will construct a family \mathcal{P}' of posets, such that

$$\mathcal{P}'_n \subseteq \mathcal{P}_n(d)$$

and

$$\lim_{n \rightarrow \infty} \frac{\log |\mathcal{P}'_n|}{n^2} = \lim_{n \rightarrow \infty} \frac{\log |\mathcal{P}_n(d)|}{n^2} = c(d).$$

A family \mathcal{P}' satisfying the above properties will be called *d-significant*.

In order to describe significant families we generalize the notion of 3-layer posets to k -layer posets in the obvious way. A partial order $P = (X, P)$ is a

k -layer partial order, if there exists a partition of its point set $X = X_1 \cup \dots \cup X_k$ into k disjoint antichains (the so-called layers) such that

$$\begin{aligned} x < y \text{ with } x \in X_i \text{ and } y \in X_j &\implies i < j, \\ \text{for every } i, j \text{ with } j > i + 1 : \quad x \in X_i, y \in X_j &\implies x < y. \end{aligned}$$

For some constants $\lambda_1, \dots, \lambda_k$ with $0 < \lambda_i < 1$ and $\sum_i \lambda_i = 1$ and a constant $0 \leq p \leq 1$, we say that a partial order P has k -configuration $Q = (\lambda_1, \dots, \lambda_k; p)$, if it belongs to the following set $\mathcal{P}(Q)$, which is defined as the set containing all k -layer partial orders in \mathcal{P} that have $p|X_i||X_{i+1}|$ comparable pairs between X_i and X_{i+1} (for all $i \in [k-1]$) and satisfy $|X_i| = \lambda_i \cdot n$ (for all $i \in [k]$).

Obviously, any two partial orders with the same k -configuration $Q = (\lambda_1, \dots, \lambda_k; p)$ must have the same number of comparable pairs, in other words for every Q there exists a $d := d(Q)$ so that $\mathcal{P}_n(Q) \subseteq \mathcal{P}_n(d)$. On the other hand, we can give an estimate for $|\mathcal{P}_{n,Q}|$, because the only degree of freedom one has when constructing a partial order in $\mathcal{P}_{n,Q}$ lies in the placement of the comparable pairs between successive layers. So it will be easy to compute $c(Q) := \lim_n \log |\mathcal{P}_{n,Q}|/n^2$.

Our aim is as follows: *for every d find a configuration Q^* such that $\mathcal{P}(Q^*)$ is d -significant*. Then compute $c(d)$ via $c(d) = c(Q^*)$. Two illustrations might help at this point: consider the configurations

$$Q_1 := \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right) \quad \text{and} \quad Q_2 := \left(\frac{1}{4}, \frac{1}{2}, \frac{1}{4}, \frac{1}{2}\right)$$

Computing $d(Q_1) = \frac{1}{8}$ and $d(Q_2) = \frac{3}{16}$ as well as $c(Q_1) = c(Q_2) = \frac{1}{4}$, we know by (16) that Q_1 is $\frac{1}{8}$ -significant and Q_2 is $\frac{3}{16}$ -significant.

Let us give a rough sketch how the above aim is achieved. Suppose d is given. First we show that there is a relatively small family of configurations \mathcal{Q} such that $d(Q') = d$ for every $Q' \in \mathcal{Q}$ and so that for every poset $P \in \mathcal{P}_n(d)$ there exists a configuration $Q' \in \mathcal{Q}$ and a poset $P' \in \mathcal{P}_n(Q')$ where P and P' differ in at most $o(n^2)$ comparable pairs. What this means is that we can actually obtain the whole of $\mathcal{P}_n(d)$ by taking the $\mathcal{P}_n(Q')$, each with a small ϵ -environment, for all $Q' \in \mathcal{Q}$. Given the relatively small size of \mathcal{Q} , this implies that there must be a d -significant configuration Q^* . It seems that we have almost got what we wanted, but: in order to compute $c(d) = c(Q^*)$, we actually need to *find* Q^* !!

Suppose that we search among all k -configurations Q with $d(Q) = d$: a necessary condition for Q^* to be d -significant is that $c(Q^*)$ must be maximal among these configurations. So, roughly speaking, our approach is as follows: for each k and d we obtain a tentative configuration $Q_{k,d}$ by solving an optimization problem. We then compare those with the same value of d and identify the one maximizing $c(Q)$. Rather than giving a pile of formulae, the result is probably best described in Figure 3.

What is most interesting about this result is the fact that there really are intervals where $c(d)$ is linear in d (represented by the dashed line). Within these intervals, the d -significant configuration has the striking feature that, as d increases, the edge density p remains constant and a new layer starts to grow.

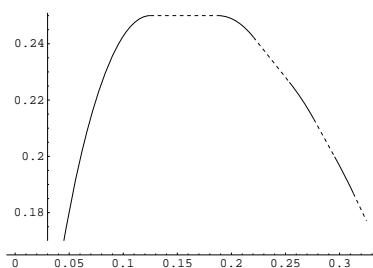


Fig. 3. $c(d)$ in the interval $d \in [0.05, 0.32]$.

This explains why these intervals are described as the *phase transitions* in the evolution of partial orders.

Unfortunately the methods used to determine $c(d)$ don't seem to be strong enough to give *almost-all*-type results for $\mathcal{P}_n(d)$. We conjecture that in fact almost all posets in $\mathcal{P}_n(d)$ lie in the d -significant configuration. Such a result would of course be nice on its own, but, more than that, it would give immediate access to the evolution of dimension, i.e. the dimension of a random partial order from $\mathcal{P}_n(d)$.

References

1. D. Achlioptas and E. Friedgut, A sharp threshold for k -colorability, *Random Structures and Algorithms* 14, 63–70, 1999.
2. D. Achlioptas and M. Molloy, The analysis of a list-coloring algorithm on a random graph, in *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, 204–212, 1997.
3. E. Friedgut, Necessary and sufficient conditions for sharp thresholds of graph properties, and the k -SAT problem, *Journal of the AMS* 12, 1017–1054, 1999.
4. M. Ajtai, J. Komlós, and E. Szemerédi, Sorting in $C \log n$ parallel steps, *Combinatorica* 3, 1–19, 1983.
5. M.H. Albert and A.M. Frieze, Random graph orders, *Order* 6, 19–30, 1989.
6. B. Bollobás, *Random Graphs*, Academic Press, London, 1985.
7. B. Bollobás, The chromatic number of random graphs, *Combinatorica* 8, 49–55, 1988.
8. B. Bollobás and G. Brightwell, Graphs whose every transitive orientation contains almost every relation, *Israel J. Math.* 59, 112–128, 1987.
9. B. Bollobás and O.M. Riordan, Constrained graph processes, *Electronic Journal of Combinatorics* 7, R18, 2000.
10. B. Bollobás and A. Thomason, Threshold functions, *Combinatorica* 7, 35–38.
11. G. Brightwell, Models of random partial orders, in *Surveys in Combinatorics* (K. Walker, ed.), Cambridge University Press, 53–83, 1993.
12. G. Brightwell, H.J. Prömel, and A. Steger, The average number of linear extensions of a partial order, *J. Combinat. Theory (A)* 73, 193–206, 1996.
13. D. Dhar, Entropy and phase transitions in partially ordered sets, *J. Math. Phys.* 19(8), 1711–1713, 1978.

14. P. Erdős, H. Kierstead, and W.T. Trotter, The dimension of random ordered sets, *Random Structures and Algorithms* 2, 253–275, 1991.
15. P. Erdős, D.J. Kleitman, and B.L. Rothschild, Asymptotic enumeration of K_n -free graphs, in *International Colloquium on Combinatorial theory, Atti dei Convegni Lincei* 17, Vol. 2, Rome, 19–27, 1976.
16. P. Erdős and A. Rényi, On random graphs I, *Publ. Math. Debrecen* 6, 290–297, 1959.
17. P. Erdős and A. Rényi, On the evolution of random graphs, *Publ. Math. Inst. Hungar. Acad. Sci.* 5, 17–61, 1961.
18. P. Erdős and A. Rényi, On the evolution of random graphs, *Bull. Inst. Internat. Statist.* 38(4), 343–347, 1961.
19. P. Erdős and A. Rényi, On the strength of connectedness of a random graph, *Acad. Sci. Hungar.* 12, 261–267, 1961.
20. P. Erdős, S. Suen, and P. Winkler, On the size of a random maximal graph, *Random Structures and Algorithms* 6, 309–318, 1995.
21. S. Janson, T. Łuczak, and A. Ruciński, *Random Graphs*, Wiley-Interscience, New York, 2000.
22. M. Karoński: Random Graphs, in *Handbook of Combinatorics*, Volume 1, 351–380. North-Holland, Amsterdam, 1995
23. J.H. Kim, The Ramsey number $R(3, t)$ has order of magnitude $t^2 / \log t$, *Random Structures and Algorithms* 7, 173–207, 1995.
24. D.J. Kleitman and B.L. Rothschild, The number of finite topologies, *Proc. Amer. Math. Soc.* 25, 276–282, 1970.
25. D.J. Kleitman and B.L. Rothschild, Asymptotic enumeration of partial orders on a finite set, *Trans. Amer. Math. Soc.* 205, 205–220, 1975.
26. D.J. Kleitman and B.L. Rothschild, A phase transition on partial orders, *Physica* 96A, 254–259, 1979.
27. Ph.G. Kolaitis, H.J. Prömel, and B.L. Rothschild, K_{l+1} -free graphs: asymptotic structure and a 0–1 law, *Trans. Amer. Math. Soc.* 303, 637–671, 1987.
28. A.C. Kaporis, L.M. Kirousis, and Y.C. Stamatiou, A Note on the Non-Colorability Threshold of a Random Graph, *Electronic Journal of Combinatorics* 7, R29, 2000.
29. E. Łamken and B.L. Rothschild, The number of odd-cycle-free graphs, in *Finite and Infinite Sets* (A. Hajnal, L. Lovász, and V.T. Sós, eds.), Colloq. Math. Soc. János Bolyai 37, North-Holland, Amsterdam, 547–553, 1984.
30. T. Łuczak, The chromatic number of sparse random graphs, *Combinatorica* 11, 45–54, 1991.
31. T. Łuczak, On triangle-free random graphs, *Random Structures and Algorithms* 16, 260–276, 2000.
32. D. Osthus, H.J. Prömel, and A. Taraz, On the evolution of triangle-free graphs, to appear in *Combinatorica*.
33. D. Osthus and A. Taraz, Random maximal H -free graphs, *Random Structures and Algorithms* 18, 61–82, 2001.
34. B. Pittel and J. H. Spencer and N. C. Wormald, Sudden emergence of a giant k -core in a random graph, *Journal of Comb. Theory Series B* 67, 111–151, 1996.
35. H.J. Prömel, T. Schickinger, and A. Steger, A note on triangle-free and bipartite graphs, to appear in *Discrete Mathematics*.
36. H.J. Prömel, T. Schickinger, and A. Steger, On the structure of clique-free graphs, to appear in *Random Structures and Algorithms*.
37. H.J. Prömel and A. Steger, The asymptotic number of graphs not containing a fixed color-critical subgraph, *Combinatorica* 12, 463–473, 1992.

- 38. H.J. Prömel and A. Steger, On the asymptotic structure of sparse triangle-free graphs, *Journal of Graph Theory* 21, 137–151, 1996.
- 39. H.J. Prömel, A. Steger, and A. Taraz, Counting partial orders with a fixed number of comparable pairs, to appear in *Combinatorics, Probability and Computing*.
- 40. H.J. Prömel, A. Steger, and A. Taraz, Phase transitions in the evolution of partial orders, *Journal of Combinatorial Theory, Series A* 94, 230–275, 2001.
- 41. A. Ruciński and N. C. Wormald, Random graph processes with degree restrictions, *Combinatorics, Probability and Computing* 1, 169–180, 1992.
- 42. A. Steger, unpublished manuscript, Technische Universität München, 1999.
- 43. W.T. Trotter, Applications of the probabilistic method to partially ordered sets, in *The mathematics of Paul Erdős II* (R.L. Graham and J. Nešetřil, eds.), Algorithms and Combinatorics, Vol. 14, Springer-Verlag, 214–228, 1997.
- 44. P. Winkler, Random structures and 0-1 laws, in *Finite and Infinite Combinatorics of Sets and Logic* (N.W. Sauer, R.E. Woodrow, and B. Sands, eds.) NATO ASI Series, Kluwer Academic publishers, Dordrecht, 1991.

Division-Free Algorithms for the Determinant and the Pfaffian: Algebraic and Combinatorial Approaches

Günter Rote

Institut für Informatik, Freie Universität Berlin, Takustraße 9, D-14195 Berlin
rote@inf.fu-berlin.de

1 Introduction

The most common algorithm for computing the determinant of an $n \times n$ matrix A is Gaussian elimination and needs $O(n^3)$ additions, subtractions, multiplications, and divisions. On the other hand, the explicit definition of the determinant as the sum of $n!$ products,

$$\det A = \sum_{\pi} \text{sign } \pi \cdot a_{1\pi(1)} a_{2\pi(2)} \cdots a_{n\pi(n)} \quad (1)$$

shows that the determinant can be computed *without* divisions. The summation is taken over the set of all permutations π of n elements. Avoiding divisions seems attractive when working over a commutative ring which is not a field, for example when the entries are integers, polynomials, or rational or even more complicated expressions. Such determinants arise in combinatorial problems, see [11].

We will describe an $O(n^4)$ algorithm that works without divisions, and we will look at this algorithm from a combinatorial and an algebraic viewpoint.

We will also consider the *Pfaffian* of a skew-symmetric matrix, a quantity closely related to the determinant. The results are in many ways analogous to those for the determinant, but the algebraic aspect of the algorithms is not explored yet.

This survey is yet another article which highlights the close connection between linear algebra and cycles, paths, and matchings in graphs [1,17,25]. Much of this material is based on the papers of Mahajan and Vinay [14,15] and of Mahajan, Subramanya, and Vinay [13].

2 Algorithms for the Determinant

2.1 Alternate Expressions for the Determinant

A direct evaluation of the expression (1) requires $n!(n-1)$ multiplications (plus a smaller number of additions and subtractions, which will be ignored in the sequel). One can try to rewrite this expression and factorize it in a clever way such that common subexpressions need to be evaluated only once. Expanding

one row or columns leads to a recursive procedure which takes $T(n) = n \cdot (1 + T(n-1)) \approx n!(e-1)$ multiplications. Splitting the matrix into two equal parts and applying Laplace expansion needs $B(n) = \binom{n}{\lfloor n/2 \rfloor} (1 + B(\lfloor n/2 \rfloor) + B(\lceil n/2 \rceil))$ multiplications. This is between 2^n (for $n > 3$) and 2^{2n} , which is still exponential.

It will turn out that one has to first *extend* the expression by additional redundant terms. Only this will allow to *reduce* the time to evaluate the expression.

2.2 Cycle Decompositions

We first rewrite the product $a_{1\pi(1)}a_{2\pi(2)} \dots a_{n\pi(n)}$ in terms of the *cycle structure* of the permutation π . Every permutation corresponds to a partition of the set $\{1, \dots, n\}$ into disjoint cycles. For example, the cycle decomposition

$$\mathcal{C} = (8 \ 7 \ 4)(6)(5 \ 3 \ 1 \ 2) = (\underline{1} \ 2 \ 5 \ 3)(\underline{4} \ 8 \ 7)(\underline{6}) \quad (2)$$

corresponds to the following permutation.

$$\pi = \begin{pmatrix} i : & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \pi(i) : & 2 & 5 & 1 & 8 & 3 & 6 & 4 & 7 \end{pmatrix}$$

We can get a canonical form of a cycle decomposition \mathcal{C} by letting each cycle start at its smallest element, which is called the *head* of the cycle and is underlined in (2), and by rearranging the cycles in increasing order of their heads. Formally, a *cycle* (c_1, c_2, \dots, c_i) of length i can be defined as a sequence of $i \geq 1$ *distinct* elements from the ground set $\{1, \dots, n\}$, where c_1 is the smallest element and is called the *head* of \mathcal{C} . The *weight* of \mathcal{C} is $\text{weight}(\mathcal{C}) = a_{c_1 c_2} a_{c_2 c_3} \dots a_{c_i c_1}$. This is the product of the weight of its arcs if \mathcal{C} is regarded as a cycle in a graph.

A *cycle decomposition* \mathcal{C} is a sequence C_1, \dots, C_k of *disjoint* cycles with increasing heads, covering every element $1, \dots, n$ exactly once. The *weight* of \mathcal{C} is the product of the weights of its cycles, and the *sign* of a cycle decomposition \mathcal{C} with k cycles is defined as

$$\text{sign } \mathcal{C} = \text{sign } \pi := (-1)^{n+k}.$$

One can check that this coincides with the usual definition of the sign and parity of a permutation.

So we can rewrite (1) as follows:

$$\det A = \sum_{\text{cycle decompositions } \mathcal{C}} \text{sign } \mathcal{C} \cdot \text{weight}(\mathcal{C}) \quad (3)$$

2.3 Clow Sequences

If we relax the requirement that all elements of a cycle are distinct, we arrive at the concept of a *closed ordered walk* or *clow* for short, in the terminology of Mahajan and Vinay [14]. A *clow* (c_1, c_2, \dots, c_i) of length i is a sequence of $i \geq 1$

numbers where the *head* c_1 of C is the unique smallest element: $c_1 < c_2, \dots, c_i$. The *weight* of C is again $\text{weight}(C) = a_{c_1 c_2} a_{c_2 c_3} \dots a_{c_i c_1}$. (Valiant [24], who introduced this concept, called this a c_1 -loop.)

A *clow sequence* \mathcal{C} is a sequence C_1, \dots, C_k of clows with a strictly increasing sequence of heads:

$$\mathcal{C} = (\underline{c_1}, \dots), (\underline{c_2}, \dots), \dots, (\underline{c_k}, \dots), \quad (4)$$

with $c_1 < c_2 < \dots < c_k$. The weight of \mathcal{C} is the product of the weights of its clows, its length is the sum of the lengths of the clows, and the sign of \mathcal{C} is just $(-1)^{n+k}$. It is clear that a cycle decomposition is a special case of a clow sequence for which the notions of weight and sign coincide with the one given above.

We can replace the summation over the set of cycle decompositions in formula (3) by the summation over the larger class of clow sequences.

Theorem 1.

$$\det A = \sum_{\text{clow sequences } \mathcal{C} \text{ of length } n} \text{sign } \mathcal{C} \cdot \text{weight}(\mathcal{C}) \quad (5)$$

Proof. It must be shown that all “bad” clow sequences which are *not* cycle decompositions cancel because the corresponding weights occur equally often with a positive and a negative sign. This is done by pairing up the bad clow sequences such that clow sequences which are matched have opposite signs. This proof technique is typical of many combinatorial proof for theorems in linear algebra [20,18,23,25]. The following lemma is stated in Valiant [24]; its proof is given in full detail in [14,15].

Lemma 1. *Let \mathcal{C} be a clow sequence that contains repeated elements. Then we can find a clow sequence $\bar{\mathcal{C}}$ with the same weight and opposite sign. Moreover, this mapping between \mathcal{C} and $\bar{\mathcal{C}}$ is an involution, i. e., if we construct the corresponding other clow sequence for $\bar{\mathcal{C}}$, we obtain \mathcal{C} again.*

Proof. Let $\mathcal{C} = C_1, \dots, C_k$. We successively add the clows C_k, C_{k-1}, \dots until a repetition occurs. Assume that C_{m+1}, \dots, C_k is a set of disjoint cycles but C_m, C_{m+1}, \dots, C_k contains a repeated element, where $1 \leq m \leq k$. Let $C_m = (c_1, \dots, c_i)$, and let c_j be the first element in this clow which is either (A) equal to a previous element c_ℓ in the clow ($1 < \ell < j$) or (B) equal to an element in one of the cycles C_{m+1}, \dots, C_k . Precisely one of these two cases will occur.

In case (A), we remove the cycle $(c_\ell = c_j, c_{\ell+1}, \dots, c_{j-1})$ from the clow C_m and turn it into a separate cycle, which we insert it into \mathcal{C} in the proper position after cyclically shifting its head to the front. In case (B), we insert the cycle into the clow C_m after the element c_j . One easily checks that the operations in cases (a) and (b) are inverses of each other; they change the number of clows by one, and hence they invert the sign. This concludes the proof of the lemma and of Theorem 1. \square

2.4 Dynamic Programming According to Length

In some sense, (1) and (3) are the most economical formulas for the determinant because all superfluous terms have been canceled and no further reduction is possible. However, we shall now see that the redundant form (5) is more amenable to an efficient algorithm. The idea is to construct clow sequences incrementally, arc by arc: We use dynamic programming to systematically compute the following quantities:

$[l, c, c_0, s]$ is the sum of all *partial clow sequences* with length l , ending in the current vertex c , where the head of the current clow is c_0 , and the sign $s = \pm 1$ gives the parity of the number of finished clows so far.

A partial clow sequence is an initial piece of the sequence given in (4): It consists of a number of completed clows, and one incomplete clow. For completing the clow sequence, we only have to know the head and the current last vertex of the incomplete clow. In the end, we need to know the parity of the total number of clows; and therefore we have to store $(-1)^k$, where k is the number of completed clows so far.

A dynamic programming algorithm can best be described by a dynamic programming graph. Our graph has $O(n^3)$ nodes corresponding to the quantities $[l, c, c_0, s]$, with $1 \leq l \leq n$, $1 \leq c_0 \leq c \leq n$, and $s = \pm 1$.

A partial clow sequence can be grown either by extending the current clow or by closing the current clow and starting a new one. Correspondingly, the dynamic programming graph has two kinds of outgoing arcs from the node $[l, c, c_0, s]$: arcs to nodes $[l + 1, c', c_0, s]$ of cost $a_{cc'}$, for all $c' > c_0$, and arcs of cost a_{cc_0} , to all nodes $[l + 1, c'_0, c'_0, -s]$, for all $c'_0 > c_0$. The latter type of arcs will start a new clow with head c'_0 and no edges yet. We consider all nodes $[0, c_0, c_0, 1]$ as start nodes and add two artificial end nodes $[n, n + 1, n + 1, s]$ with $s = \pm 1$. The sum of all path lengths from start nodes to end nodes, taken with appropriate sign according to s , is the desired quantity in (5), i. e., the determinant.

The sum of all path weights in this acyclic graph can be computed in time proportional to the number of arcs of this graph, which is $O(n^4)$: there are $O(n)$ arcs leaving each of $O(n^3)$ nodes. For example, one can calculate node values in the order of increasing l values. The storage requirement in this case is $O(n^2)$ because only the nodes with two adjacent l values need to be stored at any time.

Mahajan and Vinay [14] used this algorithm to show that the computation of the determinant belongs to the complexity class GapL, a complexity class in which summation over path weights in an acyclic graphs can be done, but, very loosely speaking, subtraction is allowed only once at the very end. One can simplify the algorithm by collapsing corresponding nodes $[l, c, c_0, +1]$ and $[l, c, c_0, -1]$ with opposite sign fields into one node $[l, c, c_0]$ and introducing “negative” arc weights: the second type of arcs have then cost $-a_{cc_0}$ instead of a_{cc_0} .

2.5 Adding a Vertex at a Time: Combinatorial Approach

The previous algorithm considers all partial clow sequences in order of increasing length l . A different factorization is possible: we can first consider the clow with

head $c_1 = 1$, if such a clow is present, and the remaining clow sequence, which is a sequence on the ground set $\{2, \dots, n\}$. For expressing the clows with head 1 we partition the matrix A into its first row r , its first column s , and the remaining matrix M .

$$A = \left(\begin{array}{c|c} a_{11} & r \\ \hline s & M \end{array} \right)$$

The sum of all clows of length l can be written as follows:

$$\begin{aligned} \text{length } l = 0: & 1 \\ \text{length } l = 1: & a_{11} \\ \text{length } l = 2: & \sum_{i=2}^n a_{1i} a_{i1} = rs \\ \text{length } l = 3: & \sum_{i=2}^n \sum_{j=2}^n a_{1i} a_{ij} a_{j1} = rMs \\ & \vdots \\ \text{length } l: & rM^{l-2}s \end{aligned}$$

The term 1 for length $l = 0$ corresponds to the “empty” clow consisting of no edges. It accounts for the case when the head of the first clow is larger than 1. We denote by q'_i the sum of signed weights of all clow-sequences of length $n-1-i$ which *don't contain* the element 1. Here we define the sign of a clow sequence with k clows as $(-1)^{n-1+k}$. Then we can write:

$$\det A = -1 \cdot q'_{-1} + a_{11} \cdot q'_0 + rs \cdot q'_1 + \dots + rM^{i-1}s \cdot q'_i + \dots \quad (6)$$

The minus sign accounts for the change of the sign convention when going from q'_{-1} over a ground set of $n-1$ elements to the original ground set with n elements. In the other terms this minus sign is canceled because we have one additional clow.

By Lemma 1, the quantity q'_{-1} must be zero: q'_{-1} is the signed sum of all clow sequences of length n on the ground set $\{2, \dots, n\}$. Such sequences must contain repeated elements and hence all terms cancel. So we get

$$\det A = a_{11} \cdot q'_0 + rs \cdot q'_1 + \dots + rM^{i-1}s \cdot q'_i + \dots + rM^{n-2}s \cdot q'_{n-1}.$$

We intend to compute the values q'_i recursively, so we need to consider clow sequences of all possible lengths. Let us denote by q_i the sum of signed weights of all clow-sequences of length $n-i$ (with the original definition of the sign). So $q_0 = \det A$, and $q'_0 = \det M$. By the above argument, we get the following formula for computing $q_0, q_1, q_2, \dots, q_n$ from the sequence $q'_0, q'_1, \dots, q'_{n-1}$.

$$\begin{pmatrix} q_n \\ q_{n-1} \\ q_{n-2} \\ \vdots \\ q_2 \\ q_1 \\ q_0 \end{pmatrix} = \begin{pmatrix} -1 & 0 & \cdots & 0 & 0 & 0 \\ a_{11} & -1 & \cdots & 0 & 0 & 0 \\ rs & a_{11} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ rM^{n-4}s & rM^{n-5}s & \cdots & a_{11} & -1 & 0 \\ rM^{n-3}s & rM^{n-4}s & \cdots & rs & a_{11} & -1 \\ rM^{n-2}s & rM^{n-3}s & \cdots & rMs & rs & a_{11} \end{pmatrix} \begin{pmatrix} q'_{n-1} \\ q'_{n-2} \\ \vdots \\ q'_2 \\ q'_1 \\ q'_0 \end{pmatrix} \quad (7)$$

This matrix multiplication is nothing but a convolution between the sequence q'_0, q'_1, \dots and the sequence $-1, a_{11}, rs, rMs, \dots$, which can most conveniently be expressed as a polynomial multiplication:

$$\begin{aligned} & q_n \lambda^n + q_{n-1} \lambda^{n-1} + \cdots + q_1 \lambda + q_0 \\ &= (-\lambda + a_{11} + rs\lambda^{-1} + rMs\lambda^{-2} + \cdots + rM^i s \lambda^{-i-1} + \cdots) \\ &\quad \times (q'_{n-1} \lambda^{n-1} + q'_{n-2} \lambda^{n-2} + \cdots + q'_1 \lambda + q'_0) \quad (8) \end{aligned}$$

Actually, these are not polynomials, but something like Laurent series in λ^{-1} . In explicit form, this means

$$\begin{pmatrix} q_n \\ q_{n-1} \\ q_{n-2} \\ \vdots \\ q_2 \\ q_1 \\ q_0 \\ 0 \\ 0 \\ \vdots \end{pmatrix} = \begin{pmatrix} -1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots \\ a_{11} & -1 & \cdots & 0 & 0 & 0 & 0 & \cdots \\ rs & a_{11} & \cdots & 0 & 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \\ rM^{n-4}s & rM^{n-5}s & \cdots & -1 & 0 & 0 & 0 & \cdots \\ rM^{n-3}s & rM^{n-4}s & \cdots & a_{11} & -1 & 0 & 0 & \cdots \\ rM^{n-2}s & rM^{n-3}s & \cdots & rs & a_{11} & -1 & 0 & \cdots \\ rM^{n-1}s & rM^{n-2}s & \cdots & rMs & rs & a_{11} & -1 & \cdots \\ rM^n s & rM^{n-1}s & \cdots & rM^2 s & rMs & rs & a_{11} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} q'_{n-1} \\ q'_{n-2} \\ \vdots \\ q'_1 \\ q'_0 \\ 0 \\ 0 \\ \vdots \end{pmatrix}$$

This is an extended version of (7) which reveals the “hidden parts” of (8). For example, the line corresponding to $q_{-1} = 0$ yields the relation

$$0 = rs \cdot q'_0 + rMs \cdot q'_1 + \cdots + rM^i s \cdot q'_i + \cdots + rM^{n-1} s \cdot q'_{n-1},$$

which may be an interesting identity but is of little use for computing anything.

Mahajan and Vinay [14] have analyzed the recursive algorithm based on (7) more carefully, and they have shown that the quantities q_i , after omitting irrelevant terms like q'_{-1} from (6), actually represent a subset of the clow-sequences that are characterized by the so-called prefix property.

2.6 The Characteristic Polynomial

We have associated the polynomial

$$P_A(\lambda) = q_n \lambda^n + q_{n-1} \lambda^{n-1} + \cdots + q_1 \lambda + q_0$$

to the matrix A , and we have shown how $P_A(\lambda)$ can be computed from the corresponding polynomial $P_M(\lambda)$ of the $(n-1) \times (n-1)$ submatrix M :

$$P_A(\lambda) = (-\lambda + a_{11} + rs\lambda^{-1} + rMs\lambda^{-2} + \cdots + rM^i s\lambda^{-i-1} + \cdots) \cdot P_M(\lambda) \quad (9)$$

This allows us to inductively compute polynomials for larger and larger submatrices, starting from $P_{(a_{nn})}(\lambda) = a_{nn} - \lambda$, until we finally obtain $P_A(\lambda)$ and the determinant $\det A = q_0$.

We have seen that the inductive approach of adding a vertex at a time naturally leads us to consider not only clow sequences of length n as in (5), but also clow sequences of all other lengths. We have also found it convenient to use the corresponding sums of weights as coefficients of a polynomial $P_A(\lambda)$. It turns out that this polynomial is nothing but the *characteristic polynomial* of the matrix A :

$$P_A(\lambda) := \det(A - \lambda I) = q_n \lambda^n + q_{n-1} \lambda^{n-1} + \cdots + q_1 \lambda + q_0 \quad (10)$$

The following well-known formula generalizes formula (3) for the determinant to all coefficients of the characteristic polynomial.

$$q_i = \sum_{\text{families } \mathcal{C} \text{ of disjoint cycles with length } n-i} \text{sign } \mathcal{C} \cdot \text{weight}(\mathcal{C}) \quad (11)$$

Here we consider not partitions into cycles, but families \mathcal{C} of disjoint cycles with a total of $n-i$ elements. The sign of such a family with k cycles is defined as $(-1)^{n-i+k}$. Formula (11) comes from the fact that a cycle family which covers $n-i$ vertices leaves i vertices uncovered, at which one can place the diagonal entries $-\lambda$ when the determinant of $A - \lambda I$ is expanded.

The following theorem states that the quantities q_i defined in the previous section are indeed the coefficients of the characteristic polynomial (10) as we know it. It is proved in the same way as Theorem 1, see [14].

Theorem 2. *The coefficients q_i ($i = 0, 1, \dots, n$) of the characteristic polynomial of A can be expressed as*

$$q_i = \sum_{\text{clow sequences } \mathcal{C} \text{ of length } n-i} \text{sign } \mathcal{C} \cdot \text{weight}(\mathcal{C}), \quad (12)$$

where the sign of a clow sequence of length $n-i$ with k cycles is defined as $(-1)^{n-i+k}$. \square

The dynamic programming algorithm of Section 2.4 can easily be extended to compute all coefficients q_i : Simply adding artificial terminal nodes $[n-i, n+1, n+1]$, with arcs of the correct sign, will do the job.

2.7 Adding a Vertex at a Time: Algebraic Approach

We will now give an algebraic proof of (7). The determinant of a bordered matrix

$$A = \left(\begin{array}{c|c} a_{11} & r \\ \hline s & M \end{array} \right) \quad (13)$$

can be expressed as

$$\det A = \det M \cdot (a_{11} - rM^{-1}s) \quad (14)$$

if the matrix M is invertible. This follows from the fact that $\det M$ is the cofactor of a_{11} , and hence $(A^{-1})_{11} = \det M / \det A$. The element $(1, 1)$ of the inverse A^{-1} can be determined by block Gauss-Jordan elimination of (13), starting with the lower right block M . This gives $(A^{-1})_{11} = (a_{11} - rM^{-1}s)^{-1}$ which leads to (14).

Now we wish to express $P_A(\lambda) = \det(A - \lambda I)$ in terms of $P_M(\lambda) = \det(M - \lambda I)$. Applying (14) to the matrix $A - \lambda I$ gives

$$\begin{aligned} P_A(\lambda) &= |A - \lambda I| = \begin{vmatrix} a_{11} - \lambda & r \\ s & M - \lambda I \end{vmatrix} \\ &= |M - \lambda I| \cdot (a_{11} - \lambda - r(M - \lambda I)^{-1}s) \\ &= P_M(\lambda) \cdot (a_{nn} - \lambda + r(\lambda I - M)^{-1}s) =: P_M(\lambda) \cdot F(\lambda) \end{aligned}$$

For simplifying the expression $F(\lambda)$, which is the multiplier which turns P_M into P_A , we rewrite $(\lambda I - M)^{-1}$ as follows.

$$\begin{aligned} (\lambda I - M)^{-1} &= \lambda^{-1}(I - \lambda^{-1}M)^{-1} \\ &= \lambda^{-1}(I + \lambda^{-1}M + \lambda^{-2}M^2 + \dots) \\ &= \lambda^{-1}I + \lambda^{-2}M + \lambda^{-3}M^2 + \dots \end{aligned}$$

This gives

$$\begin{aligned} F(\lambda) &= a_{11} - \lambda + r(\lambda I - M)^{-1}s \\ &= -\lambda + a_{11} + rs\lambda^{-1} + rMs\lambda^{-2} + rM^2s\lambda^{-3} + rM^3s\lambda^{-4} + \dots, \end{aligned} \quad (15)$$

which is the same factor as in (8) and (9).

Another proof of (7), based on the Cayley-Hamilton Theorem, is due to P. Beame, see [3].

2.8 Who Invented the Incremental Algorithm?

Berkowitz [3] used the algorithm based on the recursion (7) to derive a parallel procedure for evaluating the determinant. He called this algorithm *Samuelson's method*, giving a reference to Samuelson's original paper [19] and to the textbook of Faddeev and Faddeeva [8]. Valiant [24], who gave the first combinatorial interpretation of (7), referred to the algorithm as the Samuelson-Berkowitz algorithm. However, Samuelson's algorithm for computing the characteristic polynomial is an $O(n^3)$ algorithm that uses divisions. This is also the way in which it is described in [8, §45]. So the above algorithm can definitely not be attributed to Samuelson. However, a passage towards the end of Section §45 in [8] contains the system (7). Faddeyev and Faddeyeva state this as a possible stricter justification of Samuelson's method. The mention briefly that this formula follows from

the consideration of bordered determinants. This calculation has been sketched above in Section 2.7.

The fact that (7) appears in the section about Samuelson's method in the book [8] may be the reason for the erroneous attribution. It seems that the method ought to be credited to Faddeyev and Faddeyeva. However, being numerical analysts, they would probably not have considered the implied $O(n^4)$ procedure as a good "method". Berkowitz [3] was apparently the first one to use the recursive formula (7) algorithmically.

3 The Pfaffian

The determinant of a skew-symmetric matrix ($a_{ij} = -a_{ji}$) is the square of another expression, which is called the *Pfaffian* [12,16]. For example,

$$\begin{vmatrix} 0 & a_{12} & a_{13} & a_{14} \\ -a_{12} & 0 & a_{23} & a_{24} \\ -a_{13} & -a_{23} & 0 & a_{34} \\ -a_{14} & -a_{24} & -a_{34} & 0 \end{vmatrix} = (a_{12}a_{34} - a_{13}a_{24} + a_{14}a_{23})^2$$

The determinant of a skew-symmetric matrix of odd order is always 0. Formally, the Pfaffian of a skew-symmetric matrix A with an even number of rows and columns can be defined as follows:

$$\text{pf } A = \sum_{\text{perfect matchings } \mathcal{M}} \text{sign } \mathcal{M} \cdot \text{weight}(\mathcal{M}) \quad (16)$$

Here, a perfect matching \mathcal{M} with $k = n/2$ edges is written as

$$\mathcal{M} = (\underline{i_1}, \underline{j_1}), (\underline{i_2}, \underline{j_2}), \dots, (\underline{i_k}, \underline{j_k}), \quad (17)$$

where, by convention, $i_l < j_l$ for each l . The *sign* of the matching \mathcal{M} is defined as the sign of

$$\begin{pmatrix} 1 & 2 & 3 & 4 & \dots & n-1 & n \\ i_1 & j_1 & i_2 & j_2 & \dots & i_k & j_k \end{pmatrix}$$

when this is regarded as a permutation of $\{1, \dots, n\}$. The weight of \mathcal{M} is $a_{i_1 j_1} a_{i_2 j_2} \dots a_{i_k j_k}$. One can check that the sign of \mathcal{M} does not depend on the order in which the edges are given. Moreover, exchanging a pair i_l, j_l changes the sign but not the signed weight since $a_{ji} = -a_{ij}$. This means that we are in fact free to choose any convenient permutation π as a representation of a given matching \mathcal{M} .

Because of their close connection with matchings, Pfaffians are of interest in combinatorics, see [12,21]. For example, for certain graphs, including planar graphs, it is possible to count the number of perfect matchings in polynomial time by using Pfaffians. Knuth [10] gives a short history of Pfaffians, and argues that Pfaffians are in some way more fundamental than determinants, to which they are closely related (see Section 4).

The Pfaffian can be computed in $O(n^3)$ steps by an elimination procedure that is similar to Gaussian elimination for symmetric matrices. Alternatively, the square root of the determinant gives the Pfaffian up to the correct sign. Both approaches may be undesirable if divisions or square roots should be avoided.

3.1 Alternating Cycle Decompositions

The combinatorial approaches to the determinant of Section 2 can be applied to Pfaffians as well. We consider the product of the Pfaffians of two skew-symmetric matrices A and B . Expanding the definitions (16) leads to an overlay of two matchings \mathcal{M}_A and \mathcal{M}_B . The union of \mathcal{M}_A with \mathcal{M}_B decomposes into a disjoint union of *alternating cycles* with a total of n edges:

$$\mathcal{M}_A \uplus \mathcal{M}_B = \mathcal{C} = C_1 \cup C_2 \cup \dots$$

An alternating cycle is a cycle of even length whose edges alternate between \mathcal{M}_A and \mathcal{M}_B . Edges which are contained both in \mathcal{M}_A and \mathcal{M}_B must be kept separate the union $\mathcal{M}_A \uplus \mathcal{M}_B$; for each “common” edge, $\mathcal{M}_A \uplus \mathcal{M}_B$ contains two (distinguishable) parallel edges, forming an alternating cycle of length 2.

An alternating cycle can be traversed in two orientations. We specify the orientation by taking the vertex with the smallest number as the “first vertex” or *head* of the cycle and insisting that the first edge should belong to \mathcal{M}_A .

Taking the union of two matchings brings us back into the realm of cycle decompositions which we studied in Section 2. Now, the weight of an alternating cycle $C = (c_1, c_2, \dots, c_i)$ with $c_1 < c_2, \dots, c_i$ is

$$\text{weight}(C) = a_{c_1 c_2} b_{c_2 c_3} a_{c_3 c_4} b_{c_4 c_5} \dots a_{c_{i-1} c_i} b_{c_i c_1}. \quad (18)$$

As in Section 2.2, an *alternating cycle family* \mathcal{C} is a set of disjoint alternating cycles. Its *length* is the total number of elements in its cycles. If the length is n , we speak of an alternating cycle *decomposition*. The *weight* of \mathcal{C} is the product of the weights of its cycles, and the sign of an alternating cycle decomposition \mathcal{C} with k cycles is defined as $\text{sign } \mathcal{C} = (-1)^k$.

Theorem 3.

$$\text{pf } A \cdot \text{pf } B = \sum_{\text{alternating cycle decompositions } \mathcal{C}} \text{sign } \mathcal{C} \cdot \text{weight}(\mathcal{C})$$

Proof. The Pfaffian has been defined as a weighted sum of matchings; so the left side is a sum over all *pairs* of matchings $(\mathcal{M}_A, \mathcal{M}_B)$, where \mathcal{M}_A is regarded as a matching for A and \mathcal{M}_B is regarded as a matching for B . Now, there is a unique correspondence between pairs of matchings and alternating cycle decompositions. The only thing that has to be checked is that the signs are correct. Suppose that $\mathcal{M}_A \uplus \mathcal{M}_B$ decomposes into k disjoint cycles $\mathcal{C} = (c_1, c_2, \dots, c_i), (d_1, d_2, \dots, d_j), \dots$. We may conveniently represent \mathcal{M}_A and \mathcal{M}_B by the permutations

$$\pi_A = \left(\begin{array}{cccc|cccc|cccc} 1 & 2 & 3 & 4 & \cdots & i-1 & i & & i+1 & i+2 & \cdots & i+j & & i+j+1 & \cdots & n \\ c_1 & c_2 & c_3 & c_4 & \cdots & c_{i-1} & c_i & & d_1 & d_2 & \cdots & d_j & & & & \end{array} \right)$$

and

$$\pi_B = \left(\begin{array}{cccc|cccc|cccc} 1 & 2 & 3 & 4 & \cdots & i-1 & i & & i+1 & i+2 & \cdots & i+j & & i+j+1 & \cdots & n \\ c_2 & c_3 & c_4 & c_5 & \cdots & c_i & c_1 & & d_2 & d_3 & \cdots & d_1 & & & \cdots & \end{array} \right),$$

respectively. This gives $\text{sign } \mathcal{M}_A \cdot \text{sign } \mathcal{M}_B = \text{sign } \pi_A \cdot \text{sign } \pi_B = \text{sign}((\pi_A)^{-1} \circ \pi_B) = (-1)^k$, since the permutation $(\pi_A)^{-1} \circ \pi_B$ decomposes into k even-length cycles. \square

As an easy consequence, we obtain the relation between the determinant and the Pfaffian that was mentioned at the beginning of this section.

Corollary 1.

$$\det A = (\text{pf } A)^2$$

Proof. When we set $A = B$ in Theorem 3, we almost get the determinant formula (3): the difference between alternating cycle decompositions and cycle decompositions disappears except for the requirement that an alternating cycle must always have even length.

Changing the orientation of an odd cycle in the expansion of the determinant of a skew-symmetric matrix reverses the weight to its negative. Thus it is easy to establish a sign-reversing bijection among all cycle decompositions containing odd cycles, for example by reversing the odd cycle with the largest head. So the cycle decompositions containing odd cycles cancel. \square

If we want to use Theorem 3 to get an expression for the Pfaffian of a single matrix A , we must set B to a matrix B_0 whose Pfaffian is 1. So we select the skew-symmetric matrix

$$B_0 = \begin{pmatrix} 0 & 1 & & & & \\ -1 & 0 & & & & \\ & & 0 & 1 & & \\ & & -1 & 0 & & \\ & & & & \ddots & \\ & & & & & 0 & 1 \\ & & & & & -1 & 0 \end{pmatrix} \quad (19)$$

that represents the matching $\mathcal{M}_0 = (1, 2), (3, 4), \dots, (n-1, n)$, which we use as a *reference matching*. So we immediately get the following alternate expression for the Pfaffian.

Corollary 2.

$$\text{pf } A = \sum_{\mathcal{M}_0\text{-alternating cycle decompositions } \mathcal{C}} \text{sign } \mathcal{C} \cdot \text{weight}(\mathcal{C}) \quad \square$$

Here, the general concepts of alternating cycles, cycles families, etc., specialize as follows: An \mathcal{M}_0 -alternating cycle (c_1, c_2, \dots, c_i) is a cycle of even length i . In addition, we must have $c_3 = c_2 - 1$ if c_2 is even and $c_3 = c_2 + 1$ if c_2 is odd, and the same relation holds between c_4 and c_5 , \dots , c_i and c_1 . Its weight is given by (18) with respect to B_0 , i. e., $b_{2j-1, 2j} = 1$ and $b_{2j, 2j-1} = -1$. All other notions are unchanged.

3.2 Alternating Clow Sequences

Generalizing Theorem 1 in Section 2.3, Mahajan, Subramanya, and Vinay [13] have extended the notion of an alternating cycle cover to an *alternating clow sequence*. This gives the following theorem.

Theorem 4.

$$\text{pf } A \cdot \text{pf } B = \sum_{\text{alternating clow sequences } \mathcal{C} \text{ of length } n} \text{sign } \mathcal{C} \cdot \text{weight}(\mathcal{C}) \quad (20)$$

$$\text{pf } A = \sum_{\mathcal{M}_0\text{-alternating clow sequences } \mathcal{C} \text{ of length } n} \text{sign } \mathcal{C} \cdot \text{weight}(\mathcal{C}) \quad (21)$$

Here, an *alternating clow* (c_1, c_2, \dots, c_i) of length i is defined like a clow, with the additional restriction that i must be even. The condition on the clow head is unchanged: $c_1 < c_2, \dots, c_i$. The weight of alternating clows is defined like in (18) for alternating cycles. An \mathcal{M}_0 -*alternating clow* (c_1, c_2, \dots, c_i) must have every other edge $(c_2, c_3), (c_4, c_5), \dots, (c_i, c_1)$ belonging to \mathcal{M}_0 . Alternating clow *sequences*, their length, weight, and sign are defined just as for clows and alternating cycle families. (Our notation and terminology differs from [13].)

The proof of (20), and of (21) which is a special case of it, follows the lines of the proof of Lemma 1: a sign-reversing bijection among “bad” alternating clow sequences which contain repeated elements is established. The only change occurs in case (A), when a clow C_k forms a sub-cycle $(c_\ell = c_j, c_{\ell+1}, \dots, c_{j-1})$. If that sub-cycle has odd length, we cannot split it from the current clow because it does not form an alternating cycle. In this case, we simply reverse the orientation and replace this part of C_k by $(c_\ell = c_j, c_{j-1}, \dots, c_{\ell+1})$. The weight of C_k changes sign, and it can also be checked easily that the resulting mapping between bad alternating clow sequences remains an involution. \square

Mahajan, Subramanya, and Vinay [13] considered only the case (21) of a single matrix, and they gave a slightly different proof.

3.3 Incremental Algorithms for the Pfaffian

A dynamic programming algorithm for evaluating (21) by summing over all partial alternating clow sequences in order of increasing length can easily be formulated in analogy to Section 2.4, see [13]. However, we shall instead formulate another algorithm along the lines of the recursive algorithm based on (7) in Section 2.5.

Since Pfaffians only make sense for matrices of even order, we partition the given skew-symmetric matrix by splitting two rows and columns from it.

$$A = \left(\begin{array}{cc|c} 0 & a_{12} & r \\ -a_{12} & 0 & -s^T \\ \hline -r^T & s & M \end{array} \right)$$

Enumeration of alternating cflow sequences with head $c_1 = 1$ according to length yields the power series

$$G(\lambda) = -\lambda^2 + a_{12} + rB_0s\lambda^{-2} + rB_0MB_0s\lambda^{-4} + rB_0MB_0MB_0s\lambda^{-6} + \dots \quad (22)$$

B_0 is the “skew-symmetric unit matrix” (19) of appropriate dimension, here $(n-2) \times (n-2)$. The coefficient of λ^{-i+2} , for $i \geq 2$, represents the negative sum of all weights of all alternating cflow sequences with head 1 and length i . Note that a cflow starting with vertex 1 must terminate with the edge $(2, 1)$, and hence the factor $b_{21} = -1$ is always present in the above terms. We define the *Pfaffian-characteristic polynomial* of a skew-symmetric $n \times n$ matrix A as

$$\tilde{P}_A(\lambda) := \tilde{q}_n\lambda^n + \tilde{q}_{n-2}\lambda^{n-2} + \dots + \tilde{q}_2\lambda^2 + \tilde{q}_0,$$

where the quantities \tilde{q}_i denote the sum of signed weights of all \mathcal{M}_0 -alternating cflow-sequences of length $n-i$. This definition can also be used for odd n . Similarly as in Theorem 2, \tilde{q}_i is equal to the sum of signed weights of all \mathcal{M}_0 -alternating cycle decompositions of length $n-i$. It can also be interpreted the sum of signed weights of certain matchings with $(n-i)/2$ edges as in (16), with an appropriate definition of signs.

Note that an alternating cflow cannot have head 2. So we split alternating cflow sequences into the (possibly empty) cflow with head 1 and the remaining cflow sequence, which consists of elements $\{3, \dots, n\}$ only, in analogy to Section 2.5. We obtain the recursive equation

$$\tilde{P}_A(\lambda) = G(\lambda) \cdot \tilde{P}_M(\lambda), \quad (23)$$

which immediately leads to a recursive division-free algorithm for computing $\tilde{P}_A(\lambda)$ and the Pfaffian $\text{pf } A = \tilde{q}_0$.

4 Open Questions

The relation

$$\det A = \text{pf} \begin{pmatrix} 0 & A \\ -A^T & 0 \end{pmatrix}$$

shows that determinants are just special cases of Pfaffians that correspond to bipartite graphs. In this sense, the Pfaffian is a more basic notion, which would deserve a more thorough understanding, despite the traditional prevalence of the determinant in the curriculum and in applications. We mention a few questions that arise from the results which we have surveyed.

4.1 The Pfaffian-Characteristic Polynomial: Algebraic Interpretation

As in the case of the determinant, our algorithm has naturally lead to a polynomial $\tilde{P}_A(\lambda)$ of degree n that is associated to a skew-symmetric matrix A of

order n . In contrast to the case of the determinant, we have no idea what an algebraic interpretation of this polynomial might be, or how one might prove the relation (23) by algebraic means. It is not generally the case that the characteristic polynomial is the square of $\tilde{P}_A(\sqrt{(\lambda)})$, as the relation $q_0 = \tilde{q}_0^2$ between their constant terms would suggest.

In Theorems 3 and 4, it has turned out that it appears more natural to deal with the product of Pfaffians of *two* different matrices A and B : after all, Corollary 2 is only a special case of Theorem 3 which is by no means easier to formulate, and the same relation holds between (20) and (21) in Theorem 4. Many identities about Pfaffians involve products of two Pfaffians that are taken from submatrices of a single given matrix. One might speculate that taking two different matrices and their submatrices could lead to more general theorems and at the same time more transparent proofs in some other cases as well.

Instead of substituting the matrix B_0 from (19) into (20) of Theorem 4, one could also use the matrix

$$B_1 = \begin{pmatrix} 0 & 1 & 1 & 1 & \cdots & 1 \\ -1 & 0 & 1 & 1 & \cdots & 1 \\ -1 & -1 & 0 & 1 & \cdots & 1 \\ -1 & -1 & -1 & 0 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & -1 & -1 & \cdots & 0 \end{pmatrix},$$

which might be called the skew-symmetric analog of the all-ones matrix and has $\text{pf } B_1 = 1$. This results in a different expression for $\text{pf } A$ than (21) in Theorem 4 and to another algorithm. That algorithm is in a way simpler than the incremental algorithm given in Section 3.3, since it increments the matrix by one row and column at a time instead of two. However, it uses more computations. The matrix B_1 also leads to a different concept of a Pfaffian-characteristic polynomial.

There is an analogous algorithm for directly computing the product of two Pfaffians according to equation (20) of Theorem 4. It seems that the Pfaffian-characteristic polynomial ought to be associated to two matrices instead of one. The incremental algorithm corresponding to equation (20) leads to a polynomial $\tilde{P}_{A,B}(\lambda)$ that is related to the characteristic polynomial by the equation

$$\tilde{P}_{A,A}(\lambda) = P_A(\lambda),$$

for skew-symmetric matrices A .

Is there a way to introduce a second matrix B also into the computation of determinants in the same natural way? This might be related to the *generalized eigenvalue problem* that requires the solution of the characteristic equation

$$\det(A - \lambda B) = 0,$$

for an arbitrary matrix B instead of the usual identity matrix. The above characteristic equation is equal to the characteristic equation of AB^{-1} if B is invertible. This problem is important for some applications. Can any of the algorithms of Section 2 be adapted to compute the coefficients of this polynomial?

4.2 Complexity

All division-free methods for computing the determinant or the Pfaffian that we have described take $O(n^4)$ steps. This has already been mentioned for the dynamic programming algorithm of Section 2.4. In the recursive algorithms (7) and (23) of Sections 2.5 and 3.3, the step of the recursion from the $(n-1) \times (n-1)$ matrix M to the $n \times n$ matrix A takes $O(n^3)$ time for computing the coefficients of the factor $F(\lambda)$ or $G(\lambda)$ respectively, and $O(n^2)$ steps for multiplying the two polynomials, if this is done in the straightforward way. This amounts to a total of $O(n^4)$ steps.

The bottleneck is thus the computation of the iterated matrix-vector products in (15) and (22): for $F(\lambda)$ in (15), we successively compute the row vectors r, rM, rM^2, \dots , and multiply these by the column vector s . When the matrix A is sparse, this computation will be faster: If A has m nonzero elements, each matrix-vector product will take only $O(m)$ steps instead of $O(n^2)$ (assuming $m \geq n$). This leads to an algorithm with a running time of $O(mn^2)$. Thus, when m is small, this algorithm may be preferable to ordinary Gaussian elimination even when divisions are not an issue. Due to its simplicity, the iterative algorithm gains an additional advantage over Gaussian elimination or other direct methods, which must be careful about the choice of pivots in order to avoid numerical problems or to exploit sparsity.

Sparsity will also improve the dynamic programming algorithm of Section 2.4 from $O(n^4)$ to $O(mn^2)$ steps. It is a challenging problem to devise a division-free $O(n^3)$ method for computing the determinant.

The algorithms discussed in this paper can also be used to design good parallel algorithms for determinants and Pfaffians [3,14,13].

Despite their advantages, the methods described in this article are not well-known. A straightforward implementation of the incremental algorithm of Section 2.5 in MAPLE started to beat the built-in MAPLE procedure for the characteristic polynomial for random integer matrices of order 20, and was clearly faster already for matrices of order 40. For a simple 5×5 matrix with rational entries like $(a_{ij}) = (\frac{1}{x_i + x_j})$, our algorithm for the characteristic polynomial was two times faster than MAPLE's algorithm for computing the determinant only.

4.3 Alternative Algorithms

In the introduction it was mentioned that division-free algorithms might be useful for evaluating determinants of integers. The most straightforward algorithm would be to carry out Gaussian elimination with rational arithmetic. This approach may result in an explosive growth in the lengths of the numbers unless fractions are reduced after every step. However, to avoid the excessive growth of numbers, it is not necessary to reduce fractions by their greatest common divisor: one can perform pivots involving divisions for which the result is known to be integral. This form of integral pivoting is attributed to Camille Jordan [5, p. 69], see also [6,2,9,7]. Edmonds [6] has shown that the numbers do not explode

and hence, this algorithm runs in polynomial time. The same approach can be extended to Pfaffians [9].

Thus, for evaluating determinants or Pfaffians of integers, this approach is superior to our division-free algorithms because it takes only $O(n^3)$ operations. However, when dealing with multivariate polynomials, even “integer division” can be awkward, and division-free algorithms may prove to be worthwhile.

Strassen [22] has given a general recipe for converting an algorithm that uses divisions to obtain an integral final result into an algorithm without divisions. Divisions are simulated by expanding expressions into truncated power series. This is somehow reminiscent of the recursion (9) which also involves power series that can be truncated because the result is a polynomial. According to [4], Strassen’s approach leads to an algorithm with $O(n^5)$ steps in the case of Gaussian elimination. It would be interesting to see if there is any relation of this algorithm to the algorithms presented here.

References

1. Martin Aigner, Lattice paths and determinants. In: *Computational Discrete Mathematics*, ed. Helmut Alt, (this volume), Lecture Notes Comput. Sci., Vol. 2122 2001, pp. 1–12.
2. Erwin H. Bareiss, Sylvester’s identity and multistep integer-preserving Gaussian elimination. *Math. Comput.* **22** (1968), 565–578.
3. Stuart J. Berkowitz, On computing the determinant in small parallel time using a small number of processors. *Inf. Process. Lett.* **18** (1984), 147–150.
4. Allan Borodin, Joachim von zur Gathen, John Hopcroft, Fast parallel matrix and GCD computations. *Inf. Control* **52** (1982), 241–256
5. E. Durant, *Solution numérique des équations algébriques, tome II: systèmes des plusieurs équations*. Masson & Cie., Paris 1961.
6. Jack Edmonds, Systems of distinct representatives and linear algebra. *J. Res. Nat. Bur. Standards, Sect. B* **71** (1967), 241–245.
7. Jack Edmonds, J.-F. Maurras, Note sur les Q -matrices d’Edmonds. *RAIRO, Rech. opér.* **31** (1997), 203–209.
8. D. K. Faddeyev, V. N. Faddeyeva, *Vyčislitel’nye metody lineĭnoĭ algebrы* (in Russian), Moscow, 1960. English translation: D. K. Faddeev, V. N. Faddeeva, *Numerical Methods of Linear Algebra*. Freeman, San Francisco 1963. German translation: D. K. Faddejewa, W. N. Faddejewa, *Numerische Methoden der linearen Algebra*, several editions since 1964.
9. G. Galbiati and Francesco Maffioli, On the computation of pfaffians. *Discr. Appl. Math.* **51** (1994), 269–275.
10. Donald E. Knuth, Overlapping Pfaffians. *Electron. J. Comb.* **3** (1996), No. 2, article R5, 13 pp. Printed version: *J. Comb.* **3** (1996), No. 2, 147–159.
11. Christian Krattenthaler, Advanced determinant calculus. *Séminaire Lotharingien de Combinatoire* **B42q** (1999), 67 pp.
12. László Lovász, M. D. Plummer, *Matching Theory*. Ann. Discr. Math., Vol. 29. North-Holland Mathematics Studies, Vol. 121. Amsterdam 1986.
13. Meena Bhaskar Mahajan, P R Subramanya, V Vinay, A combinatorial algorithm for Pfaffians. In: *Computing and combinatorics. Proc. 5th annual international conference. (COCOON ’99), Tokyo, July 1999*, ed. Takao Asano et al., Lecture

- Notes Comput. Sci. 1627, Springer-Verlag, pp. 134–143 (1999). Extended version: DIMACS Technical Report 99-39, Rutgers University, July 1999.
14. Meena Bhaskar Mahajan, V Vinay, Determinant: Combinatorics, algorithms, and complexity. *Chicago J. Theor. Comput. Sci.*, Vol. 1997, Article no. 1997-5, 26 pp.
 15. Meena Mahajan, V. Vinay, Determinant: Old algorithms, new insights. *SIAM J. Discrete Math.* **12** (1999), 474–490.
 16. Thomas Muir, *A Treatise on the Theory of Determinants*. MacMillan and Co., London 1882; repr. Dover, New York 1960.
 17. Günter Rote, Path problems in graphs. In: *Computational graph theory*, ed. Gottfried Tinhofer et al., Computing Suppl. **7**, 155–189, Springer-Verlag, Wien 1990.
 18. D. E. Rutherford, The Cayley-Hamilton theorem for semi-rings. *Proc. Roy. Soc. Edinburgh*, Sect. A **66** (1961–64), 211–215 (1964).
 19. Paul A. Samuelson, A method of determining explicitly the coefficients of the characteristic equation. *Ann. Math. Statist.* **13** (1942), 424–429.
 20. Dennis Stanton, Dennis White, *Constructive combinatorics*. Springer-Verlag, New York 1986.
 21. John R. Stembridge, Nonintersecting paths, pfaffians, and plane partitions. *Adv. Math.* **83** (1990), 96–113.
 22. Volker Strassen, Vermeidung von Divisionen. *J. reine angew. Math.* **264** (1973), 184–202.
 23. Howard Straubing, A combinatorial proof of the Cayley-Hamilton theorem. *Discrete Math.* **43** (1983), 273–279.
 24. Leslie G. Valiant, Why is Boolean complexity theory difficult? In: *Boolean Function Complexity*, ed. M. S. Paterson, LMS Lecture Notes Series, Vol. 169, Cambridge Univ. Press, 1992, pp. 84–94.
 25. Doron Zeilberger, A combinatorial approach to matrix algebra. *Discrete Math.* **56** (1985), 61–72.

Check Character Systems and Anti-symmetric Mappings[★]

Ralph-Hardo Schulz

Department of Mathematics and Computer Science, Free University of Berlin
Arnimallee 3, 14195 Berlin, Germany
`schulz@math.fu-berlin.de`

1 Introduction

1.1 First Definitions and Historical Remarks

A *check digit system* with one check character over an alphabet A is a code

$$c : \begin{cases} A^{n-1} \longrightarrow A^n \\ a_1 a_2 \dots a_{n-1} \longmapsto a_1 a_2 \dots a_{n-1} a_n. \end{cases}$$

which is used to detect (but not in general to correct) single errors (i.e. errors in one component) and other errors of certain patterns (discussed below).

Historically, among the first publications¹ are articles by FRIEDMAN & MENDELSON (1932; cf. [9]) based on code-tables (after an International Telegraph conference) and by Rudolf SCHAUFFLER (1956; cf. [19]) using algebraic structures. In his book VERHOEFF (1969; cf. [27]) presented basic results which are in use up to the present time.

1.2 Error Types to Be Detected

Which types of errors (of human operators) have to be detected ? This question was answered more or less by statistical sampling made by VERHOEFF in a Dutch postal office and by BECKLEY, see Table 1. They show that single errors and *adjacent transpositions* (neighbour transpositions), i.e. errors of the form $\dots ab \dots \rightsquigarrow \dots ba \dots$, are the most prevalent ones (beside insertion and deletion errors which can be detected easily when all codewords have the same length n).

Note that the last two digits of a word may be affected by single errors more than all the other digits ([27] p. 14).

1.3 Systems over Groups

The systems most commonly in use are defined over alphabets endowed with a group structure. For a group $G = (A, \cdot)$ one can determine the check digit a_n

[★] Based on a lecture given at the graduate school on May 31, 1999, and on [24], [25].

¹ as J.Dénes found

Table 1. Error types and their frequencies

<i>Error type</i>		<i>Relative frequency</i>	
		Verhoeff	Beckley
single error	$\dots a \dots \rightsquigarrow \dots a' \dots$	79.0% (60-95)	86%
adjacent transposition	$\dots a b \dots \rightsquigarrow \dots b a \dots$	10.2 %	8%
jump transposition	$\dots acb \dots \rightsquigarrow \dots bca \dots$	0.8%	
twin error	$\dots aa \dots \rightsquigarrow \dots bb \dots$	0.6%	
phonetic error ($a \geq 2$)	$\dots a0 \dots \rightsquigarrow \dots 1a \dots$	0.5%	
jump twin error	$\dots aca \dots \rightsquigarrow \dots bcb \dots$	0.3%	
other error		8.6%	6%

Source: Verhoeff [27](12,112 pairs, 6 digits), Beckley [1].

Table 2. Detection of other errors

<i>Error type</i>	<i>Detection possible if</i>
twin errors	$xT(x) \neq yT(y)$ for all $x, y \in G$ with $x \neq y$
jump transpositions	$xyT^2(z) \neq zyT^2(x)$ for all $x, y, z \in G$ with $x \neq z$
jump twin errors	$xyT^2(x) \neq zyT^2(z)$ for all $x, y, z \in G$ with $x \neq z$

such that the following (check) equation holds (for fixed permutations δ_i of G , $i = 1, \dots, n$, and an element e of G , for instance the neutral element).

$$\delta_1(a_1)\delta_2(a_2)\dots\delta_n(a_n) = e \quad (1)$$

Such a system detects all *single errors*; and it detects all *adjacent transpositions* iff for all $x, y \in G$ with $x \neq y$

$$x \cdot \delta_{i+1}\delta_i^{-1}(y) \neq y \cdot \delta_{i+1}\delta_i^{-1}(x). \quad (2)$$

The proofs are straightforward. Often, one chooses a fixed permutation T of G and puts $\delta_i := T^i$ for $i = 1, \dots, n$. Equation (2) then becomes²

$$x T(y) \neq y T(x) \quad \text{for all } x, y \in G \text{ with } x \neq y. \quad (3)$$

A permutation T of G satisfying (3) is called **anti-symmetric**. Conditions for the detection of other errors are shown in Table 2.

1.4 First Examples

Well-known systems are

² Some authors take $a_{n-1}\dots a_1a_0$ as the codeword numeration and therefore $\phi^{n-1}(a_{n-1})\dots\phi(a_1)a_0 = e$ as the check equation. Then anti-symmetry is defined by $\phi(x)y \neq \phi(y)x$ for $x, y \in G, x \neq y$. Taking the inverse mapping T^{-1} as ϕ one can transform this condition into (3) and vice versa.



Fig. 1. Example of an EAN (with bar-code) and an ISBN.

- the **European Article Number code (EAN)** and (after adding 0 as first digit) the **Universal Product Code (UPC)** with $G = (\mathbb{Z}_{10}, +)$, $n = 13$, $e = 0$, $\delta_{2i-1}(a) = a =: L_1(a)$ and $\delta_{2i}(a) = 3a =: L_3(a)$; this system does not detect adjacent transpositions $\dots ab\dots \rightsquigarrow \dots ba\dots$ for $|a - b| = 5$: the mapping $L_3 L_1^{-1}$ is not anti-symmetric. An example of an EAN is shown in Figure 1.
- the **International Standard Book Number code (ISBN)** with $G = (\mathbb{Z}_{11}, +)$, $n = 10$, $e = 0$ and $\delta_i(a) = ia =: L_i(a)$ for $i = 1, \dots, 10$; this system detects all adjacent transpositions but needs an element $X \notin \{0, \dots, 9\}$.
- the system of the **serial numbers of German banknotes** (see e.g. [20] p.64–67.) An example of a serial number is shown in Fig. 3. (The solution for the check digit ■ is given at the end of this article.) In this system, G is D_5 , the dihedral group of order 10 (see below) and $n = 11$, $\delta_i = T_0^i$ for $i = 1, \dots, 10$ and $\delta_{11} = \text{id}$; here $T_0 = (01589427)(36)$ is an anti-symmetric permutation found by VERHOEFF (cf. [27]). Thus, the check equation is

$$T_0(a_1) * T_0^2(a_2) * \dots * T_0^{10}(a_{10}) * a_{11} = 0.$$

Letters of the serial numbers are coded as follows:

A	D	G	K	L	N	S	U	Y	Z
0	1	2	3	4	5	6	7	8	9

The *dihedral group* D_m of order $2m$ is the symmetry group of the regular m -gon. Denoting the rotation through angle $2\pi/m$ by d and a reflection by s (see Fig. 2) one has $D_m = \langle d, s \mid e = d^m = s^2 \wedge ds = sd^{-1} \rangle$. The $2m$ elements are of the form $d^i s^j$ for $i = 0, \dots, m-1$ and $j = 0, 1$. For any natural number m one can identify the element $d^i s^j \in D_m$ with the integer $i + j \cdot m$ ($i = 0, \dots, m-1$; $j = 0, 1$). Thus one obtains a representation of D_m on $\{0, \dots, 2m-1\}$; we denote the induced operation by $*$. The composition table for the case $m = 5$ is shown in Table 3.

2 Anti-symmetric Mappings

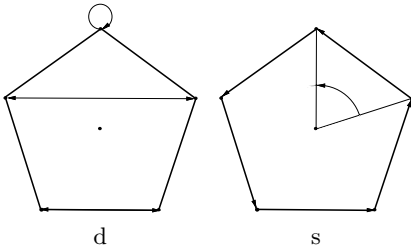
2.1 The Abelian Case

For an abelian group G , condition (3) is equivalent to

$$xT(x)^{-1} \neq yT(y)^{-1} \text{ for all } x, y \in G \text{ with } x \neq y. \quad (4)$$

Table 3. The operation on $\{0, 1, \dots, 8, 9\}$ induced by D_5 .

*	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	0	6	7	8	9	5
2	2	3	4	0	1	7	8	9	5	6
3	3	4	0	1	2	8	9	5	6	7
4	4	0	1	2	3	9	5	6	7	8
5	5	9	8	7	6	0	4	3	2	1
6	6	5	9	8	7	1	0	4	3	2
7	7	6	5	9	8	2	1	0	4	3
8	8	7	6	5	9	3	2	1	0	4
9	9	8	7	6	5	4	3	2	1	0

**Fig. 2.** Generators of D_5 (as symmetries of the regular pentagon).**Fig. 3.** What does the last digit (■) of DK9673165S■ look like?

A permutation T satisfying (4) is called an *orthomorphism* or *perfect difference mapping* and $\frac{1}{T} : x \mapsto T(x)^{-1}$ is said to be a *complete mapping*, cf. MANN (1942) [17]. The theory of complete mappings is well developed. Thus one knows for example

2.1.1 Theorem. a) A finite abelian group G admits a complete mapping iff G has odd order m or contains more than one involution; (PAIGE 1947 [18]).

b) A necessary condition for a finite group of even order to admit complete mappings is that its Sylow 2-subgroups be non-cyclic. For soluble groups this condition is also sufficient; (HALL and PAIGE 1955 [15]).

A consequence of this theorem is the following corollary for which DAMM [4] gave a short prove using groups with “sign”, i.e. with a homomorphism $G \rightarrow \{-1, +1\}$.

2.1.2 Corollary. (i) A group of order $2m$ where m is odd does not admit a complete mapping.

(ii) \mathbb{Z}_{10} does not admit a check digit system which detects all single errors and all adjacent transpositions.

(iii) Like the EAN, no other system using \mathbb{Z}_{10} is able to detect all adjacent transpositions. More generally:

(iv) A cyclic group G admits an anti-symmetric mapping iff $|G|$ is odd.

(v) Groups of order $m = 2u$ with u odd, in particular D_5 and \mathbb{Z}_{10} , do not admit a check digit system which detects all twin errors or all jump twin errors.

2.2 Further Examples

1. We mention several other anti-symmetric mappings of D_m . If m is odd then, by defining $d = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$ and $s = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$, the dihedral group D_m can be represented as a matrix group (see e.g. [12]), namely $D_m \cong \left\{ \begin{pmatrix} a & 0 \\ b & 1 \end{pmatrix} \mid a, b \in \mathbb{Z}_m \wedge a \in \{1, -1\} \right\}$. a) For m odd the mapping

$$T \begin{pmatrix} a & 0 \\ b & 1 \end{pmatrix} := \begin{pmatrix} a & 0 \\ h_a(b) & 1 \end{pmatrix}$$

is anti-symmetric if $h_a(b) = u_a - ab$ with $u_1 \neq u_{-1}$ (see [21] 3.7).

Choosing $u_a = -at - c$ with $c, t \in \mathbb{Z}_m$ and $t \neq 0$ one gets the system of GUMM ([12] p.103), namely

$$T(d^k) = d^{c+t-k} \quad \text{and} \quad T(d^j s) = d^{t-c+j} s,$$

in particular for $t = r/2 = -c$ one of VERHOEFF's anti-symmetric mappings; and putting $u_{-1} = 0$ and $u_1 = 1 - m$ (or $c = t = (m - 1)/2$ in GUMM l.c.) yields the system of BLACK ([2]) for $m = 5$ and of ECKER and POCH ([8] Th.4.4). If one puts $c = t = 1$ in GUMM's system, one gets the scheme of GALLIAN and MULLIN ([11]Th.2.1) for m odd: $T(d^k) = d^{2-k}$ and $T(d^j s) = d^j s$.

b) GALLIAN and MULLIN observed that for $m = 2k$ and $G = D_m$ the following mapping is anti-symmetric; ([11] l.c.; see as well [4] p.22).

$$\begin{aligned} T(s) &= e & T(d^{-1}s) &= ds & T(d^j) &= d^{1-j} & (k+1 \leq j \leq m) \\ T(d^j) &= d^{1-j}s & (1 \leq j \leq k) & & T(d^j s) &= d^{j+1}s & (1 \leq j \leq k-1) \\ T(d^j s) &= d^{j+1} & (k \leq j \leq m-2) & & & & \end{aligned}$$

2. Let $q = 2^m > 2$ and $K = \text{GF}(q)$; put $u_{ac} = 1$ if $a^2 \neq c$ and otherwise $u_{ac} = u$ for a fixed $u \in K \setminus \{0, 1\}$ Then the mapping

$$T : \begin{pmatrix} a & 0 \\ b & c \end{pmatrix} \mapsto \begin{pmatrix} a^2 & 0 \\ u_{ac} \cdot b & c^2 \end{pmatrix}$$

is an anti-symmetric mapping of the group

$$G_0 = \left\{ \begin{pmatrix} a & 0 \\ b & c \end{pmatrix} \mid a, b \in K \wedge a \cdot c \neq 0 \right\}$$

of all regular 2×2 - **triangular matrices** over $\text{GF}(q)$; (see [22] 3.1).

3. For $m \geq 2$, the group

$$Q_m := \langle a, b \mid a^{2m} = b^4 = e, b^2 = a^m, ab = ba^{-1} \rangle$$

is called a **dicyclic group** or (for m a power of 2) a *generalized quaternion group* and for $m = 2$ *quaternion group*; it is a group of order $4m$. One obtains an anti-symmetric mapping φ in the following way (cf. [11] Th.2.1 ii).

$$\begin{aligned}\varphi(a^i) &= a^{-i} \quad (\text{for } 0 \leq i \leq m-1) \quad \text{and} \quad \varphi(a^i) = b \cdot a^{i-1} \quad (\text{for } m \leq i \leq 2m-1) \\ \varphi(ba^i) &= ba^{i-1} \quad (\text{for } 0 \leq i \leq m-1) \quad \text{and} \quad \varphi(ba^i) = a^{-i} \quad (\text{for } m \leq i \leq 2m-1).\end{aligned}$$

4. Further examples can be found below and, for example, in [8], [22], [21], [24].

2.3 Existence Theorems

The following theorem, similar to the abelian case, has a rather technical proof:

2.3.1 Theorem (GALLIAN and MULLIN). *Let G be a group and $g \in G$. The mapping φ with $\varphi(x) = gx^{-1}$ is anti-symmetric iff g commutes with no element of order 2; (cf. [11] Th.3.1).*

An important tool for the construction of anti-symmetric mappings is the following.

2.3.2 Extension-Theorem (GALLIAN and MULLIN). *If H is a normal subgroup of G and there exist anti-symmetric mappings φ and ψ of H and G/H respectively, then there exists an anti-symmetric mapping of G ; (cf. [11]).*

Proof (Sketch). Put $\gamma(u_i h) = \varphi(h)\psi^*(u_i)$ where ψ^* is the mapping induced by ψ on a set of representatives $\{u_i\}$ of the cosets of H . \square

In particular, the direct product of groups with anti-symmetric mappings has an anti-symmetric mapping; this was already known to GUMM [12] and, implicitly, to VERHOEFF. So one can extend the results on the existence of anti-symmetric mappings from p -groups which are different from cyclic 2-groups to nilpotent groups with trivial or non-cyclic Sylow 2-subgroup. This led to the *Conjecture of Gallian and Mullin* ([11]) which has been confirmed by HEISS [13], [14]:

2.3.3 Theorem (HEISS). *Every finite non-abelian group admits an anti-symmetric mapping.*

2.4 Anti-automorphisms and Good Automorphisms

In this section we shall use automorphisms and anti-automorphisms to construct anti-symmetric mappings. We start with anti-automorphisms. The mapping $\text{inv}: x \mapsto x^{-1}$ is, under certain conditions, an anti-symmetric mapping. On the other hand, inv is, for every group, an anti-automorphism.

2.4.1 Definition. A bijection $\psi: G \rightarrow G$ of a group G is called an **anti-automorphism** if $\psi(xy) = \psi(y) \cdot \psi(x)$ for all $x, y \in G$.

The set of all anti-automorphisms of G is denoted by $\text{Antaut } G$. Note that $\text{Antaut } G = \text{Aut } G \circ \text{inv}$. DAMM uses anti-automorphisms to construct anti-symmetric mappings. He states:

2.4.2 Theorem (DAMM [4], [5]). (a) If φ is anti-symmetric and ψ an anti-automorphism then $\psi \circ \varphi^{-1} \circ \psi^{-1}$ is anti-symmetric.

(b) For an anti-automorphism ψ the following are equivalent: (i) ψ is anti-symmetric. (ii) ψ is fixed point free. (iii) $\varphi^{-1} \circ \psi \circ \varphi$ is fixed point free for any (anti-) automorphism φ .

We continue with group automorphisms.

2.4.3 Proposition. Let G be a finite group and $T \in \text{Aut } G$. Then T is anti-symmetric iff T does not fix any conjugacy class of $G \setminus \{e\}$ (where e denotes the identity element of G). When G is abelian, this is the case iff T operates fixed point freely on G ; (see [23] 3.1.)

When determining necessary and sufficient conditions for the detection of errors, one comes to the following

2.4.4 Definition. Let G be a finite group. An automorphism T of G is called **good** provided $T(x)$ is not conjugate to x or x^{-1} and $T^2(x)$ is not conjugate to x or x^{-1} for all $x \in G, x \neq e$ (cf.[3]).

2.4.5 Remarks. (i) A good automorphism is anti-symmetric and detects single errors, adjacent transpositions, jump transpositions, twin errors and jump twin errors; (see 2.4). (ii) If G is abelian then the automorphism T detects single errors, adjacent transpositions, jump transpositions and twin errors if T^2 is fixed point free; and T is good if T^4 is fixed point free.

2.4.6 An Example (cf.[3]). Choose $q = 2^m > 2$ and G as the **Sylow 2-subgroup of the unitary group** $\text{SU}(3, q^2)$ of order q^3 , formed by the matrices

$$Q(x, y) = \begin{pmatrix} 1 & x & y \\ 0 & 1 & x^q \\ 0 & 0 & 1 \end{pmatrix} \quad \text{with } x, y \in GF(q^2) \text{ and } y + y^q + x^{q+1} = 0.$$

The automorphism $T : Q(x, y) \mapsto Q(x\lambda^{2q-1}, y\lambda^{q+1})$, induced by conjugation with

$$H_\lambda = \begin{pmatrix} \lambda^{-q} & 0 & 0 \\ 0 & \lambda^{q-1} & 0 \\ 0 & 0 & \lambda \end{pmatrix}$$

for $\lambda \in GF(q^2) \setminus \{0\}$ is good iff the multiplicative order of λ is not a divisor of $q + 1$. Generalization:

2.4.7 Good Automorphisms on p -Groups. Let P be a p -group and $T \in \text{Aut } P$. Suppose $\gcd(o(T), p(p-1)) = 1$. Then T is good iff T is fixed point free on P ; (cf.[3]).

2.4.8 Corollary. Let S be the **Sylow 2-subgroup of $\text{PSL}(2, q)$** , $q = 2^m, m > 1$ defined by

$$S = \left\{ \begin{pmatrix} 1 & 0 \\ v & 1 \end{pmatrix} \mid v \in GF(q) \right\}; \quad \text{then } T = \begin{pmatrix} t & 0 \\ 0 & t^{-1} \end{pmatrix}$$

with $t \in GF(q) \setminus \{0, 1\}$ acts fixed point freely on S . Therefore S admits a good automorphism and hence a check digit system which detects all single errors,

neighbour-transpositions, twin errors, jump transpositions and jump-twin errors; (cf.[3]).

Similarly, the *Sylow 2-subgroups of the Suzuki group $\mathbf{Sz}(q)$* ($q = 2^{2t+1}, q > 2$) admit a good automorphism. More generally

2.4.9 Theorem. *The Sylow 2-subgroup of a Chevalley group over $\text{GF}(q)$, $q = 2^m$, admits a good automorphism T with $o(T) \mid (q-1)$ provided q is large enough; (cf.[3] Result 2).*

3 Equivalence of Check Digit Systems

Although the systems over Chevalley groups are able to detect all five types of prevalent errors we concentrate now on the dihedral group of order 10 since its elements can be interpreted as $0, 1, \dots, 9$ in the decimal system.

Because there are (exactly) 34,040 anti-symmetric mappings over D_5 (VERHOEFF [27] p.92, DAMM [4] p.44, GIESE [10]) we want to define equivalences between these (and the corresponding schemes). There are several possibilities to do so. Throughout this section, let G be a group and T_1, T_2 permutations of G .

3.1 Weak Equivalence

3.1.1 Definition. T_1 and T_2 (and the related schemes) are called **weakly equivalent** if there exist elements $a, b \in G$ and an automorphism $\alpha \in \text{Aut } G$ such that

$$T_2 = R_a \circ \alpha^{-1} \circ T_1 \circ \alpha \circ L_b ;$$

here $R_a(x) := x \cdot a$ and, as before, $L_b(y) := by$; (cf. [27], [4] p.38, [24]). Weak equivalence is an equivalence relation.

3.1.2 Theorem. a) *If T_1 and T_2 are weakly equivalent and if T_1 is anti-symmetric, then T_2 is also anti-symmetric ([27], [4]) .*

b) *If T_1 and T_2 are weakly equivalent permutations of G then they detect the same percentage of twin errors ([24], [10]).*

c) *If T_1 is an automorphism of G and if T_2 is weakly equivalent to T_1 then T_1 and T_2 detect the same percentage of jump transpositions and the same percentage of jump twin errors ([24], [10]).*

Proof (Sketch). b) We have (for $\bar{x} = \alpha(bx)$ and $\bar{y} = \alpha(by)$):

$$xT_2(x) \neq yT_2(y) \iff x\alpha^{-1}T_1\alpha(bx)a \neq y\alpha^{-1}T_1\alpha(by)a \iff \bar{x}T_1(\bar{x}) \neq \bar{y}T_1(\bar{y})$$

c) We get $xyT_2^2(z) \neq zyT_2^2(x) \iff \bar{x}\bar{y}T_1^2(\bar{z}) \neq \bar{z}\bar{y}T_1^2(\bar{x})$ for $\bar{x} = \alpha(bx)$, $\bar{z} = \alpha(bz)$ and $\bar{y} = \alpha(y)T_1(\alpha(b))$. Similarly for jump twin errors. \square

3.1.3 Weak Equivalence and Detection Rates. The following counter-example (cf. [10], [24]) shows that systems with weakly equivalent anti-symmetric permutations may have different detection rates. Let T_0 be the anti-symmetric mapping $T_0 = (01589427)(36)$ of VERHOEFF. It detects 94.22 % of jump transpositions and 94.22 % of jump twin errors. The weakly equivalent permutation

Table 4. Types of anti-symmetric mappings of D_5 and their detection rates

	I	IIa	IIb	III	IV	VIa/b	V
single errors	100%						
adjacent transpos.	100%						
twin errors	95.56	95.56	91.11	91.11	91.11	55.56	
jump transpositions	94.22	92	94.22	92	90.22	66.67	
jump twin errors	94.22	92	94.22	92	90.22	66.67	
Detection rate of all 5 error types (weighted)	99.90	99.87	99.87	99.84	99.82	99.30	99.85- 99.42
number of classes	2	44	8	160	16	1/5	1470
elements in a class	20	20	20	20	20	20/4	20

Source: GIESE [10],[24].

$T_1 := R_4 \circ \text{id} \circ T_0 \circ \text{id} \circ L_3$, namely $T_1 = (079482)(36)$ detects only 87.56 % of all jump transpositions and jump twin errors respectively. Therefore, we look for equivalence relations preserving the detection rates.

3.2 Automorphism Equivalence and Strong Equivalence

3.2.1 Definition. T_1 and T_2 (and the related systems) are called **automorphism equivalent** if there exists an $\alpha \in \text{Aut } G$ such that $T_2 = \alpha^{-1} \circ T_1 \circ \alpha$; and they are said to be **strongly equivalent** if they are automorphism equivalent or if there exists an anti-automorphism ψ with $T_2 = \psi^{-1} \circ T_1^{-1} \circ \psi$; ([24], [25]).

3.2.2 Proposition. *Automorphism equivalence and strong equivalence are equivalence relations; and if T_1 and T_2 are automorphism equivalent then T_1 and T_2 are weakly equivalent. If T_1 and T_2 are automorphism equivalent or strongly equivalent, then T_1 and T_2 detect the same percentage of adjacent transpositions, jump transpositions, twin errors and jump twin errors; ([10], [24]).*

3.2.3 The Dihedral Group of Order 10. a) *Types of equivalence classes.* According to computations by GIESE with the program package MAGMA there are 1,706 equivalence classes of anti-symmetric mappings with respect to automorphism equivalence [10]. S. Giese distinguishes 6 types of classes according to the rate of detection of errors, see Table 4.

b) *Some representatives.* To Type I there belong e.g. T_0 , (03986215)(47) and (07319854)(26) (VERHOEFF's mappings); the mappings of GUMM, SCHULZ, BLACK and WINTERS mentioned in 2.2 belong to Type VIb.

3.2.4 The Quaternion Group Case. By coding the elements of $Q_2 = \langle a, b | a^4 = e \wedge b^2 = a^2 \wedge ab = ba^{-1} \rangle$ by $a^i b^j \mapsto i + 4j$ ($i = 0, \dots, 3; j = 0, 1$) one gets Table 5 as the multiplication table. There exist exactly 1,152 anti-symmetric mappings of Q_2 which constitute 48 equivalence classes of size 24 each with respect to automorphism equivalence (as S. Ugan found out using C^{++}). The

Table 5. Multiplication table of the quaternion group.

*	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	0	5	6	7	4
2	2	3	0	1	6	7	4	5
3	3	0	1	2	7	4	5	6
4	4	7	6	5	2	1	0	3
5	5	4	7	6	3	2	1	0
6	6	5	4	7	0	3	2	1
7	7	6	5	4	1	0	3	2

Table 6. Multiplication in Q_3 .

$i * j$	$0 \leq j \leq 5$	$6 \leq j \leq 11$
$0 \leq i \leq 5$	$(i + j) \text{MOD } 6$	$(i + j) \text{MOD } 6 + 6$
$6 \leq i \leq 11$	$(i - j) \text{MOD } 6 + 6$	$(i - j + 3) \text{MOD } 6$

mapping of GALLIAN & MULLIN, (0)(1362745), belongs to Type I. (For more details see [26], [25]).

3.2.5 The Dicyclic Group of Order 12. (a) As defined in 2.2, Q_3 is $\langle a, b | a^6 = e \wedge b^2 = a^3 \wedge ab = ba^{-1} \rangle$. The elements of this group can be coded by the numbers 0 to 11 by $a^i b^j \mapsto i + 6j$ ($i = 0, \dots, 5$; $j = 0, 1$). This yields the multiplication shown in Table 6.

(b) According to a computer search by S. UGAN (1999)[26], see as well [25], there are exactly 1,403,136 anti-symmetric mappings of Q_3 ; (this means that only approximately 0.3% of the $12!$ permutations of Q_3 are anti-symmetric). Further results are shown in Table 7.

c) *Representatives for Type I in Q_3 .* Representatives of the 4 classes of Type I with respect to strong equivalence are (0 1 6 9 10 8 2 5 11 3 7 4), (0 2 6 11 7 9 3 8 1 5 4 10), (0 6 8 10 9 5 3 11 1 2 4 7), (0 6 1 3 4 11 8 9 7 5 2 10).

d) *The Mapping of Gallian and Mullin.* For $m = 3$, their anti-symmetric mapping is (0)(1 5 8 2 4 9 3 10 11 6 7); it has a detection rate of 81.82% for twin errors, of 92.42% for jump transpositions and jump twin errors respectively; the weighted rate for all 5 errors under consideration is 99.79% (cf. Ugan [26]).

4 Generalization to Quasigroups

$(Q, *)$ is called a **quasigroup** if the equations $x * b = c$ and $a * y = c$ have a unique solution x and y (respectively) for every $a, b, c \in Q$. Quasigroups are another way to describe **Latin squares**, cf. [6], [7], [16].

Let $(Q, *_i)$ be quasigroups; then one uses as check equation

$$(\dots (x_n *_n x_{n-1}) *_n x_{n-2}) \dots *_1 x_0 = d$$

Of importance for error detection are now (i) the anti-symmetry of $(Q, *)$:

$$x * y = y * x \implies x = y \quad (\text{for all } x, y \in Q)$$

Table 7. Types of check digit systems of Q_3 with detection rates of over 90% for each considered error type.

	Type I	Type II	Type III/IV
single errors		100%	
adjacent transpos.		100%	
twin errors	96.97%	96.97–93.94	96.97–90.91
jump transpositions	94.70%	95.71–93.94	95.96–90.15
jump twin errors	94.70%	95.45–93.18	95.96–90.15
all 5 error types (weighted)	99.92%	99.91	99.90–99.82
number of automorphism equivalence classes	8	204	804/26,464
number of strong equivalence classes	4	102	402/13,232
number of check digit systems	96	2,448	9,648/317,568

Source: Ugan [26],[25]

and (ii) the *total anti-symmetry*, that means anti-symmetry with

$$(c * x) * y = (c * y) * x \implies x = y \quad (\text{for all } x, y \in Q).$$

For more details see e.g. [8], [22],[4].

5 Solution of the Exercise

The alpha-numeric serial number of the banknote of Figure 2 with hidden check digit is DK9673165SA. Substituting the letters as indicated in section 1.4 one obtains 1396731656A. Applying the check equation gives

$$T_0(1) * T_0^2(3) * T_0^3(9) * T_0^4(6) * T_0^5(7) * T_0^6(3) * T_0^7(1) * T_0^8(6) * T_0^9(5) * T_0^{10}(6) * \mathbf{A} = 0,$$

$$\text{an equation equivalent to } \underbrace{5 * 3 * 7 * 6 * 9 * 3 * 0 * 6 * 8 * 6}_{7 \quad 1 \quad 6 \quad 6 \quad 2} * \mathbf{A} = 0;$$

$7 * 1 * 6 * 6 * 2 * \mathbf{A} = 0$ leads to $6 * 0 * 2 * \mathbf{A} = 0$ which has $\mathbf{A} = 9^{-1} = 9$ as solution.

References

1. Dudley F. Beckley. An optimum system with modulus 11. *The Computer Bulletin*, 11:213–215, 1967.
2. William L. Black. Error detection in decimal numbers. *Proc IEEE (Lett.)*, 60:331–332, 1972.
3. Claudia Broecker, Ralph-Hardo Schulz, and Gernot Stroth. Check character systems using chevalley groups. *Designs, Codes and Cryptography, DESI.*, 10:137–143, 1997.

4. Michael Damm. Prüffziffersysteme über Quasigruppen. Diplomarbeit Universität Marburg, März 1998.
5. Michael Damm. Check digit systems over groups and anti-symmetric mappings. *Archiv der Mathematik*, to appear.
6. J. Dénes and A.D. Keedwell. *Latin Squares and their Applications*. Academic Press, New York, 1974.
7. J. Dénes and A.D. Keedwell. *Latin Squares. New Developments in the Theory and Applications*. North Holland, Amsterdam, 1991.
8. A. Ecker and G. Poch. Check character systems. *Computing*, 37(4):277–301, 1986.
9. W. Friedman and C. J. Mendelsohn. *Notes on Codewords*. *Am. Math. Monthly*, pages 394–409, 1932.
10. Sabine Giese. Äquivalenz von Prüfzeichensystemen am Beispiel der Diedergruppe D_5 . Staatsexamensarbeit, FU Berlin. Jan. 1999.
11. Joseph A. Gallian and Matthew D. Mullin. Groups with antisymmetric mappings. *Arch. Math.*, 65:273–280, 1995.
12. H. Peter Gumm. A new class of check-digit methods for arbitrary number systems. *IEEE Trans. Inf. Th. IT*, 31:102–105, 1985.
13. Stefan Heiss. Antisymmetric mappings for finite solvable groups. *Arch. Math.*, 69(6):445–454, 1997.
14. Stefan Heiss. Antisymmetric mappings for finite groups. Preprint, 1999.
15. M. Hall and L.J. Paige. Complete mappings of finite groups. *Pacific J. Math.*, 5:541–549, 1955.
16. Charles F. Laywine and Gary L. Mullen. *Discrete Mathematics using Latin Squares*. J. Wiley & Sons, New York etc., 1998.
17. H.B. Mann. The construction of orthogonal latin squares. *Ann. Math. Statistics*, 13:418–423, 1942.
18. L.J. Paige. A note on finite abelian groups. *Bull. AMS*, 53:590–593, 1947.
19. R. Schauffler. Über die Bildung von Codewörtern. *Arch. Elektr. Übertragung*, 10(7):303–314, 1956.
20. R.-H. Schulz. *Codierungstheorie. Eine Einführung*. Vieweg Verlag, Braunschweig / Wiesbaden, 1991.
21. R.-H. Schulz. A note on check character systems using latin squares. *Discr. Math.*, 97:371–375, 1991.
22. R.-H. Schulz. Some check digit systems over non-abelian groups. *Mitt. der Math. Ges. Hamburg*, 12(3):819–827, 1991.
23. R.-H. Schulz. Check character systems over groups and orthogonal latin squares. *Applic. Algebra in Eng., Comm. and Computing, AAECC*, 7:125–132, 1996.
24. R.-H. Schulz. On check digit systems using anti-symmetric mappings. In I. Althöfer et al., editor. *Numbers, Information and Complexity*, pages 295–310. Kluwer Acad.Publ. Boston, 2000.
25. R.-H. Schulz. Equivalence of check digit systems over the dicyclic groups of order 8 and 12. In J. Blankenagel & W. Spiegel, editor, *Mathematikdidaktik aus Begeisterung für die Mathematik*, pages 227–237. Klett Verlag, Stuttgart, 2000.
26. Sehpahnur Ugan. Prüfzeichensysteme über dizyklischen Gruppen der Ordnung 8 und 12. Diplomarbeit, FU Berlin, Oct. 1999.
27. J. Verhoeff. *Error detecting decimal codes*, volume 29 of *Math. Centre Tracts*. Math. Centrum Amsterdam, 1969.

Algorithms in Pure Mathematics

Gernot Stroth

FB Mathematik und Informatik, Institut für Algebra und Geometrie,
Martin-Luther-Universität Halle Wittenberg, 06099 Halle
`stroth@coxeter.mathematik.uni-halle.de`

1 Introduction

In this article, we will discuss algorithmic group theory from the point of view of pure mathematics. We will investigate some of the problems and show why many problems are accessible just for a few years. In algebra there are many problems, which are easy to state, sometimes even easy to prove, and where one might be surprised that there is no algorithmic solution. The two most developed areas of algebra in the sense of existence of algorithms are number theory and group theory. As the understanding of the algorithms in number theory usually needs a lot of deep theory I will restrict myself mainly to group theory. We will look at the algorithms by themselves and not which one is better to implement and similar questions. So this paper will be also interesting for all those who will probably never use a computer to solve a problem. We will see that to find algorithms for basic or more advanced questions is a deep mathematical problem, which uses many results in pure mathematics, and even more important, raises new questions in pure mathematics. So we can say that looking for algorithms is part of pure mathematics, moreover it is a highly nontrivial part.

Basically we are not so interested in the result as in the proof, and the proof has to be an algorithmic one. Now as mathematicians of course we are used that for some results there is more than one proof. Any mathematician has an individual feeling which proof might be better or more interesting. Hence also there might be more than one algorithm to solve a problem. So what we need is a criterion to compare algorithms. Here is a first try to do it in a mathematical manner:

Algorithm A is better than algorithm B, if there is a natural number n , such that for all up to finitely many input data the algorithm A is at least n -times faster than algorithm B

In the application of course the number of input data is always finite. The same applies for what is widely used as asymptotical running time, depending on the length m of the input data. Then it is said that the algorithm A is better than B if the quotient of the asymptotical running time $TA(m)/TB(m)$ is convergent to 0 with m to infinity. But this also means that maybe in reality A is better than B if m becomes so large, that we could not put such an object on any existing computer in the world. So both are not definitions for all days use. Furthermore both definitions are dangerous as A might be better than B but in

practice B is always better than A . But from the point of view of mathematics we could live with either of both. We are free of the type of the computer used and free of all tricks implementing the algorithm. There is no fear that A is better than B today but this might change in 20 years. Things like this should not occur in mathematics. The message is that we have to give up the idea that we could compare algorithms in practice. The definition of *better* should not be related to any application right now. This method of mathematics is the most fruitful on the long run and has shown the best applicability anyway. But we have to be a little bit more careful. Before we could give the definition above, we had to define *algorithm* and *running time*. This we will avoid in this article. The main reason for this is that there is no easy definition. There are definitions via recursive sets and Turing machines or universal computers. But it is not possible really to work with these definitions.

Now we are in a critical situation. We do not know what is meant by algorithm, running time, and so we cannot compare algorithms. On the other hand everybody has a certain intuition what an algorithm might be. We will live with this naive intuition in this article. We should remember that we do great parts of mathematics without really to go deep into axiomatic set theory. A strict definition we will just need if we like to prove theorems of type

Algorithms with certain properties do not exist

There are results of this type. The negative solution of Hilbert's 10th problem is such one.

Is there an algorithm to decide the solvability of diophantine equations?

The answer to this question as is well known is no.

But we will not speak about results like this in this article.

With all these problems in mind, we still have to ask ourselves which algorithms do we search for. Here we will take a definition which is widely used in theoretical computer science. We will assume that the input data are given in bits. If the running time could be given by a polynomial in these bits, we will call the algorithm good or polynomial. This is just a time complexity we do not care about memory complexity. For the remainder of this article we will look for polynomial algorithms.

Of course our input data will not be bits, we have fields and groups and so on. We will not be interested in the realization of these objects inside the computer, in practice this could play a role. Our results will be of the form

For this problem there is a good algorithm

Never we will have a result of type: For this problem there is no good algorithm, as we have no exact definition for many things which would be needed for such a result. Further even if we know that there is a good algorithm we will never ask the question about the nature of the corresponding polynomial or even

which might be the one with the smallest degree. These question can depend on the particular realization of the group or field.

Here one example that shows that the question above might have very deep answers.

Let K be an algebraic number field, \mathcal{O} be the algebraic integers in K and \mathcal{O}^* be the units. Let Δ be the discriminant. Then there is a good algorithm which determines a generator of \mathcal{O}^* and calculates the class group. If n is the degree there is an upper bound for the running time given by

$$(2 + \log|\Delta|)^{o(n)} |\Delta|^{\frac{3}{4}}$$

Now here is a question

Is there an algorithm for which we may replace $\frac{3}{4}$ by $\frac{1}{2}$?

This is true if the Riemannian conjecture is true. Obviously we see that for dealing with algorithms in algebra one needs a certain amount of background knowledge. This we will develop in the next section as far as it is necessary for the problems we will deal with lateron.

One final remark. When we speak about algorithms in this lecture, we will always mean deterministic algorithms. There are also quite a lot of probabilistic algorithms which came up during the last years. These, at least as far as group theory is concerned, do even more use the classification of the finite simple groups and very detailed knowledge of the particular groups. Hence the group theoretical background is much more involved than with the algorithms we are going to present in this paper.

2 Galois Groups

We will deal with abstract groups and with Galois groups. Both concepts are very close together. Groups themselves are the mathematical abstraction of symmetry. To study groups in a certain sense is the same as to study symmetries in the world. We will restrict ourselves to finite groups, i.e. objects with finitely many symmetries. The modern group theory basically was born with the work of E. Galois. Of fundamental importance was his discovery of nonabelian finite simple groups on May 30, 1832. As it is known Galois was asking whether we can write the zeros of polynomials just by the use of arithmetic operations and radicals as it is true for polynomials of degree at most 4. It was known that this is not true, in particular there is a polynomial of degree 5 where it is false. Now Galois asked the question why. The fundamental new idea in the work of Galois was to attach to any polynomial equation $f = 0$ a group G . In a modern language the following happens. Let a_1, \dots, a_n be the zeros in \mathcal{C} and $K = \mathbb{Q}(a_1, \dots, a_n)$. Then let G be the automorphism group of K . Now the group G has subgroups U and Galois found that there is a bijective relation between the subgroups of G and the fields L between \mathbb{Q} and K . He asked the question whether one can

find a polynomial g (then of smaller degree) which in the sense above belong to some extension $\mathbb{Q} \subseteq L$, i.e. L is generated by the roots of g . In general this is not the case. The subgroups which have this property we call nowadays normal subgroups. Then for a normal subgroup N the corresponding field L has the shape $L = \mathbb{Q}(b_1, \dots, b_m)$, where b_i are the zeros of some polynomial equation $g = 0$. The corresponding group now is the factor group G/N , which usually is smaller than G and we have opened the door to induction proofs. The groups one really has to work with are those which besides 1 and G do not have any other normal subgroup. These groups we call simple. Now in general for any group G we have a sequence

$$1 = K_0 \trianglelefteq K_1 \trianglelefteq \dots \trianglelefteq K_r = G$$

where K_i/K_{i-1} is simple for $i = 1, \dots, r$. Hence the question about solvability of equations is reduced to the case of equations with simple groups. Galois proved that an equation is solvable in the sense described above if and only if all the simple factors in G are cyclic of prime order. These groups we also call solvable. But not just because it solved a deep problem also from a general algebraic point of view the work of Galois was a breakthrough. Obviously the simple groups are the building blocks of the finite groups (like primes for numbers). Galois recognized this important property of simple groups. He also proved the first classification theorem.

Let G be a finite non abelian simple group, $|G| \leq 60$, then $G \cong A_5$.

This was the beginning of the classification of the finite simple groups which finally was established in 1981. The finite simple groups at the turn of 19th century were A_n and $PSL(n, q)$. Further in 1860 and 1873 E. Mathieu [16], [17], [18] discovered five simple groups M_{11} , M_{12} , M_{22} , M_{23} and M_{24} (the index i stands for the smallest i such that M_i is a subgroup of A_i , the alternating group of degree i). These groups did not fit to any of the known series at that time, so Burnside [1] in his group theory book called them sporadic. At the end of the classification we should know that there are 26 sporadic groups. A further breakthrough was the classification of the simple Lie algebras over \mathbb{C} due to Cartan and Killing [2]. To these algebras there belong groups, the so called Lie groups. The classification of the algebras yields series and 5 exceptional algebras. For any of the series there are finite analogues of the corresponding Lie groups, i.e. $PSL(n, q)$, $Sp(2n, q)$, $O(n, q)$. To the five exceptional algebras we have the Lie groups $G_2(\mathbb{C})$, $E_6(\mathbb{C})$, $E_7(\mathbb{C})$, $E_8(\mathbb{C})$ and $F_4(\mathbb{C})$. In 1901 Dickson [6] was able to find the analogue $G_2(q)$ and more or less $E_6(q)$. But he was not able to continue. The next milestone was the discovery of Chevalley that the Lie algebras possess a basis such that all structure constants are integers. Hence one can read them over any field. Now the procedure which gets the Lie groups out of the algebras gets over finite fields the finite analogues. These groups nowadays are called the Chevalley groups [3], [4]. Lateron Steinberg studied the procedure which gives us the unitary group $GU(n, q)$ out of $GL(n, q^2)$ and showed that this can be generalized in certain circumstances. These groups are called the

Steinberg groups [25]. Finally Suzuki and Ree showed that for $Sp(4, 2^n)$, $G_2(3^n)$ and $F_4(2^n)$, n odd, there is a variant of Steinberg's construction, which also leads to three new series [26], [22], [23]. These are the series of finite simple groups. Between 1963 and 1976 there were 21 further sporadic simple groups discovered. The classification of the finite simple groups says that a finite simple group is cyclic of prime order, one member of the series just described, or one of the 26 sporadics. So in 1981 one of the fundamental questions of finite group theory, the question which are the building blocks, was solved.

Parallel to the classification there was a development of algorithms in group theory which solved the everyday problems. But these algorithms used a very restricted theoretical background. After the classification we have much better theoretical knowledge and so one can search for good algorithms. In this context we will investigate two areas, elementary group theory and the determination of Galois groups.

Let us start with the Galois groups. Given an equation how can we determine their Galois group. Here it turns out that it is not enough just to work over \mathbb{Q} . As we have seen before there are intermediate fields L to consider, i.e. extensions $L \leq K$. Hence the problem reads as follows

Let K be an algebraic number field, $f \in K[x]$. Determine G_f . Is there a good algorithm?

Theorem 1. *Let $b \in \mathbb{N}$. Then there is a good algorithm which decides whether $|G_f| \leq b$. If this is true it will end with G_f .*

Remark. Unfortunately the polynomial which describes the running time depends on b .

Proof. This algorithm is known to anybody from the algebra course. First we factorize f into irreducible factors in $K[x]$. How to do this see [5], [11], [12] and [13]. Let g be a nonlinear irreducible factor. Define $L = K[x]/gK[x]$. Now apply the algorithm to L . This algorithm determines the splitting field of f . If we find some intermediate field whose degree over K is larger than b , so $|G_f| > b$ and we stop. If not, then for the splitting field M we have $[M : K] \leq b$. Now we have to determine the Galois group. For this we have to find a primitive element α and then m_α . Now we have G . How to do this see [20].

Let $n = \deg f$, then we know $|G| \leq n!$. Hence for bounded n there is always a good algorithm. This is a typical example of a theoretical good result which has nothing to do with what is needed in practice. In fact the situation is that we have for any n a special algorithm. The question the Galois theory wanted to solve first was the question whether G_f is solvable.

Theorem 2. *There is a good algorithm which decides whether G_f is solvable.*

Proof. We may assume that f is irreducible. If $|G_f| \leq p(n)$ for some polynomial p , then we would be done. Hence the question is, is there a polynomial p such that for any solvable transitive group G of degree n we have $|G| \leq p(n)$. Unfortunately this is not true. Let for example $n = 2^k$. Let G be a Sylow 2-subgroup of the symmetric group Σ_n , then $|G| = 2^{n-1}$, so the order of G is not bounded by a polynomial in n . But for primitive permutation groups there is such a bound.

A group is called primitive if the stabilizer of a point is a maximal subgroup (or equivalently if G leaves no nontrivial partition of the set $\{1, \dots, n\}$ invariant). There is a bound due to Palfy [19] which is $\frac{1}{\sqrt[3]{2n}}n^c$, with $c \sim 3.25$. (see also [21])

The main theorem of Galois theory implies that G_f is primitive is equivalent to:

$$\text{If } K \subseteq L \subseteq K(\alpha), \text{ with } f(\alpha) = 0, \text{ then } L = K \text{ or } L = K(\alpha).$$

The idea now is as follows. We construct a tower of fields

$$K = K_0 \subset K_1 \subset \dots \subset K_t = K(\alpha)$$

which cannot be refined, i.e. to $K_i \subset K_{i+1}$ belongs a primitive group. Now we apply Theorem 1. Hence we just have to solve the following problem. If $K \subset K(\alpha)$ find L with $K \subseteq L \subseteq K(\alpha)$ and L maximal. This can be achieved as follows. We factorize f over $K(\alpha)$. Let g be an irreducible factor $g \neq x - \alpha$. We define a field $L_g \neq K(\alpha)$. If $g = x - \beta$, there is an automorphism σ of $K(\alpha)$ with $\sigma(\alpha) = \beta$. Now let L_g be the fixed field of σ . If H is the fixed group belonging to $K(\alpha)$ then $\langle H, \sigma \rangle$ belongs to L_g . If g is not linear, then let β be some zero of g in some extension field of $K(\alpha)$. Now set $L_g = K(\alpha) \cap K(\beta)$. The group belonging to L_g is $\langle H, H^\sigma \rangle$. Galois theory tells us that the L_g contain all maximal subfields of $K(\alpha)$. Now choose L_g with $[L_g : K]$ maximal. This is our L .

Remark. The algorithm above does not provide us with G . We get the order and all composition factors, so we can decide whether the group is solvable or not.

By the classification of the finite simple groups we do know the structure of all primitive permutation groups (Scott-O'Nan - Theorem [14]). So it should be possible to continue as before. We will see how this can be done later on. For primitive groups we have the following result. If G is primitive of degree n and there is no regular normal subgroup in G then $|G| < n^{c \log \log n}$, with $c < 10$.

The problem in the determination of Galois groups seems to come from the large groups. But this is not really true.

Theorem 3. *There is a good algorithm which decides whether $G_f \cong A_n$ or Σ_n .*

Proof. Without loss we may assume $n \geq 8$. By the classification of the finite simple groups the only 6-fold transitive groups are A_n and Σ_n . Now we go on as in theorem 1. If after the first 6 steps we have a field of degree $n(n-1)(n-2)(n-3)(n-4)(n-5)$, then G is 6-fold transitive and so $G \cong A_n$ or Σ_n . To distinguish which case holds, we just have to calculate the discriminant.

The classification of the simple groups even provides us with a classification of the 2-fold transitive groups. This can be decided after the second step. But then we have a large variety of groups and to find out the right one seems to be the problem. There is a good algorithm which determines the socle of G , i.e. the product of all minimal normal subgroups of G . To determine G itself seems to be a hard problem.

Remark. For almost all n we have: If G is primitive of degree n then $G \cong A_n$ or Σ_n .

But now we will move to the elementary group theory.

3 Groups

We will return to Galois again. If we have a group G with $|G| < 60$, then G is solvable. This nowadays is an exercise in any introductory course in group theory. But how do we do it? Let for example $|G| = 36$. Then we choose a Sylow 3-subgroup S of G . By Sylow's theorem we either have exactly one or four Sylow 3-subgroups. In the former this is normal and the factorgroup is a 2-group, so G is solvable. So assume that we have exactly four Sylow 3-subgroups. Then we have an homomorphism σ from G into Σ_4 . As $|\Sigma_4| = 24$, we have $N = \ker \sigma \neq \emptyset$. Further $N \neq G$, as G permutes the four Sylow 3-subgroups transitively. Now both $|N|$ and $|G/N|$ are smaller than $|G|$ and we may proceed by induction.

The problem with this proof is, that it is not Galois' proof, as the Sylow theorem is roughly 40 years after Galois. We do not know how Galois proved this result. But anyway Sylows theorem is the most powerful tool in finite group theory.

Sylow Theorem. *Let G be a finite group $|G| = p^a n$, where p is some prime and p does not divide n . Then the following holds*

- (1) G possesses a subgroup P with $|P| = p^a$ (We call any such group a Sylow p -subgroup of G).
- (2) All Sylow p -subgroups of G are conjugate (i.e. if P_1, P_2 are Sylow p -subgroups there is some $g \in G$ with $P_1 = g^{-1}P_2g := P_2^g$.)
- (3) The number of Sylow p -subgroups of G is of the form $1 + kp$.

Because of the theoretical importance it is clear that one is looking for algorithms to find a Sylow p -subgroup. Most algorithms use one of the proofs one will see in an introductory course in group theory.

We start with any nontrivial p -subgroup P . Then we consider $N_G(P)$. If $N_G(P) = G$, then we can use induction on G/P . So assume $N_G(P) < G$. Then by induction we find a Sylow p -subgroup of $N_G(P)$. Hence we may assume that P is a Sylow p -subgroup of $N_G(P)$. Now let P act on the set $\mathcal{P} = \{P^g \mid g \in G\}$. Suppose that P has a fixed point $Q \neq P$, then PQ is a p -group and $PQ > P$. Now we start the same procedure with PQ instead of P . So we may assume that P is the only fixed point in \mathcal{P} . Then all other orbits have length divisible by p . Now we have $|\mathcal{P}| = |G : N_G(P)| = 1 + kp$. In particular P is a Sylow p -subgroup of G .

Remarks. a) For this algorithm we have to find some nontrivial p -group in the first place. This is guaranteed by Cauchy's theorem, which says that if the group order is divisible by p then G has an element of order p . The usual proof for this is not constructive. Meanwhile there is a good algorithm for this, which

uses the classification of the finite simple groups. But in the existing computer algebra programs usually such an element will be found just by accident. The probability to find an element whose order is divisible by p is fairly good.

b) That all Sylow p -subgroups are conjugate is also not constructive, hence to find g which conjugates two given Sylow p -subgroups was not touched in the proof just described.

c) The real bottleneck in the proof before is the determination of $N_G(P)$ as the following theorem shows

Theorem 4. (Luks [15]) *If there is a good algorithm, which for given p -subgroup P of G determines $N_G(P)$, then there is a good algorithm for the graph isomorphism problem.*

Most people believe that there is no good algorithm for the graph isomorphism problem, so probably there is no good algorithm for the determination of $N_G(P)$. Does this mean there is no good algorithm for finding a Sylow p -subgroup? We have to look for proofs which do not use the calculation of normalizers. Hence it might be a little bit surprising that we have the following theorem

Theorem 5. (Kantor[10]) *There is a good algorithm, which finds a Sylow p -subgroup. Moreover there is a good algorithm which finds some $g \in G$ such that $P_1^g = P_2$ if P_1, P_2 are two given Sylow p -subgroups of G .*

We will just sketch a proof. Suppose first that G is solvable. There is a good algorithm which determines G' [7], the derived subgroup. From this we can decide whether there is a normal subgroup M of index p in G . So assume first there is such a subgroup.

Step 1: Assume that P, P_0 are two Sylow p -subgroups of G with

$$P \cap M = P_0 \cap M \trianglelefteq G$$

Then find $g \in G$ with $P^g = P_0$.

Let M/N be a chief factor of G in M with $P \cap M \leq N$. Then G/N is an extension of an elementary abelian q -group, $q \neq p$, i.e. a vector space over $GF(q)$, with a cyclic group of order p , which acts irreducibly on M/N . Then using linear algebra, we can write down the element g . (Let $h \in P \setminus M$, $h_0 \in P_0 \setminus M$ such that $t = h^{-1}h_0 \in M$. Set $u = \sqrt[p]{t}$, which is possible as M/N is a p' -group. Then $g = u^h(u^2)^{h^2} \cdots (u_{p-1})^{h^{p-1}}$ does the job.)

Step 2. If P and P_0 are Sylow p -subgroups of G then there is a good algorithm which finds a $g \in G$ with $P^g = P_0$.

By induction we can find some $m \in M$ with $(P \cap M)^m = P_0 \cap M$. Set $G^* = \langle P^m, P_0 \rangle$ and $M^* = G^* \cap M$. Then as $P \cap M^*$ is of index p in both P_0 and P^m , we see that $P \cap M^* \trianglelefteq G^*$. Now we may apply step 1.

Step 3. There is a good algorithm which finds a Sylow p -subgroup.

If p does not divide $|G/G'|$, we can proceed by induction with G' . So we may assume that there is some $M \trianglelefteq G$ with $|G : M| = p$. Choose any $g \in G \setminus M$. By

induction we now can find a Sylow p -subgroup Q of M . By step 2 we can find an $m \in M$ with

$$(Q^g)^m = Q.$$

Let $\langle h \rangle$ be a Sylow p -subgroup in $\langle gm \rangle$. Then $\langle Q, h \rangle$ is a Sylow p -subgroup of G .

How do we proceed in the general case? Let us analyze the proof first. We started with an normal subgroup M such that G/M is cyclic. In the general case this will be a simple group. Then one has to be able to find a chief factor, which was step 1. So if we have a chief series of G the question of finding a Sylow p -subgroup is reduced to the question to find some in a finite simple group. Now the algorithm for the chief series exist

Theorem 6. (Ronyai[24]) *There is a good algorithm to calculate a chief series of G*

This algorithm is based on the classification of the finite simple groups and in particular uses the proof of the Schreier conjecture that the automorphism group of a finite simple group is solvable.

So it remains the problem to determine a Sylow p -subgroup in a given finite simple group. Here we will assume that our group is given as a primitive permutation group of degree n . This is not any restriction.

Theorem 7. (Kantor [8]) *Let $|G| > n^8$, then G is some classical group or A_m .*

If $G \cong A_m$, then the action is the one on a partition of $\{1, \dots, m\}$. For the alternating groups it is not a problem to write down the Sylow p -subgroups. For the classical groups there are good algorithms to find the natural representation [9], [10]. But then it is just a problem of linear algebra to determine a Sylow p -subgroup.

We see that this algorithm uses the classification of the finite simple groups very heavily. Not just that we know the list, it also uses very detailed information about the particular groups in question.

The main problem with all these algorithms is how do we recognize a simple group and how do we find a primitive representation we can work with. Hence this identification problem is a highly nontrivial one and many people work on it.

The algorithm for solvable groups works very effectively. The one for nonsolvable groups has this bound n^8 , which can be improved but it just means that for a certain number of groups we have to work by hand. So for example for the monster we have the order $|M| = 2^{46} \cdot 3^{20} \cdot 5^9 \cdot 7^6 \cdot 11^2 \cdot 13^3 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 \cdot 41 \cdot 47 \cdot 59 \cdot 71$, which is roughly 10^{53} . The smallest permutation representation is of degree roughly 10^{20} , or $E_8(q)$, whose order in a polynomial of degree 248 in q , whence $|E_8(2)| \sim 10^{74}$.

As said in the beginning all the results we showed are of theoretical interest, in practice there are better algorithm, which of course are not polynomial. If we look at the manual for MAGMA, a group theory computer algebra package, we find: The determination of centralizers and intersections is cheap (i.e. fast and

efficient). The calculation of Sylow p -subgroups is in the middle area. The test of simplicity is expensive.

The theoretical situation is: For test of simplicity there is a polynomial algorithm, for Sylow p -subgroups we just saw one. For the calculation of centralizers and intersections there is none, this is related to the $P \neq NP$ -problem.

But also for very simple and in application quite often raised questions there are now good algorithms known.

- 1) Determine the Frattini subgroup of a group, i.e. the intersection of all maximal subgroups. The problem is that we have to calculate intersections. On the other hand these elements are exactly those which can be dropped in any generating system for G .
- 2) Let G be a p -group. determine the Thompson subgroup $J(G)$. This is the subgroup of G which is generated by the abelian subgroups of maximal order.

We have seen that algorithms may be a part of pure mathematics. It is not to prove theorems with restricted tools. It is not the constructive mathematics from the early 20th century. We use every result available. It is like any mathematics to gather knowledge. The main question is: Is there a good algorithm, we do not have really to implement some. But, as said in the introduction, the problem is that we do not really know what good means. Is polynomial really the right definition? Maybe it is one of the big challenges to find right definitions, not just for good. Mathematics always was created starting with some weak definitions, working with them until some point where they proved not to be sufficient. Also in that sense group theory is a good example. The associative law came 50 years after Galois. Maybe some day we will really speak about a theory of algorithms which starts with easy understandable and widely accepted axioms.

References

1. W. Burnside, Theory of finite groups, 2nd. edn. Cambridge 1911; Dover Publications, 1955
2. E. Cartan, Oeuvres Complètes I-1, Paris, Gauthier-Villars, 1952.
3. C. Chevalley, Sur certains groupes simple, Tohoku Math. J. 7, 1955, 14–66.
4. C. Chevalley, Seminaire Chevalley, Classification des Groupes de Lie Algebriques, Vol. 2, Paris 1956–58.
5. A. L. Chistov, D. Yu. Grigoryev, Polynomial time factoring of the multivariable polynomials over a global field, LOMI preprint E5–82, Leningrad 1982.
6. L. Dickson, A class of groups in an arbitrary realm connected with the configuration of the 27 lines in a cubic surface, J. Math. 33, 1901, 145–173.
7. M. Furst, J. Hopcroft, E. Luks, Polynomial-time algorithms for permutation groups, Proc. 21st IEEE Symposium Foundations of Computer Science, 1980, 36–41.
8. W. Kantor, Permutation representations of the finite classical groups of small degree or rank, J. Algebra 60, 1979, 158–168.
9. W. Kantor, Polynomial-time algorithms for finding elements of prime order and Sylow subgroups, J. Algorithms 6, 1985, 478–514.

10. W. Kantor, Sylow's theorem in polynomial time, *J. Comput. Syst. Sci.* 30, 1985, 359–394.
11. S. Landau, Factoring polynomials over algebraic number fields, *SIAM J. on Comp.* 14, 1985, 184–195.
12. A.K. Lenstra, Factoring polynomials over algebraic number fields, *Lecture Notes of Computer Science* 162 (Proc. of EUROCAL), Springer 1983, 245–254.
13. A.K. Lenstra, H.W. Lenstra Jr., L. Lovasz, Factoring polynomials with rational coefficients, *Math. Annalen* 261, 1982, 515–534.
14. M. Liebeck, C.E. Praeger, J. Saxl, On the O'Nan–Scott theorem for finite primitive permutation groups, *J. Australian Math. Soc.* 44, 1988, 389–396.
15. E.M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, *J. Comput. Syst. Sci.* 25 (1982), 42–65.
16. E. Mathieu, Memoire sur le nombre de valeurs que peut acquerir une fonction quand on y permut ses variables de toutes les manieres possible, *Liouville's J.* 5, 1860, 9–42.
17. E. Mathieu, Memoire sur l'etudes des fonctions de plusieurs quantites sur la maniere de les formes et sur les substitutions qui les laissent invariables, *Liouville's J.* 6, 1861, 241–323.
18. E. Mathieu, Sur la fonction cinq fois transitive des 24 quantites, *Liouville's J.* 18, 1873, 25–46.
19. J. Palfy, A polynomial bound for the orders of primitive solvable groups, *J. Alg.* 77 (1982), 127–137.
20. M. Pohst, H. Zassenhaus, *Algorithmic algebraic number theory*, Cambridge Univ. Press, 1989.
21. L. Pyber, A. Shalev, Asymptotic results for primitive permutation groups, *J. Alg.* 188, 1997, 103–124.
22. R. Ree, A family of simple groups associated with the simple Lie algebra F_4 , *Amer. J. Math.* 83, 1961, 401–420.
23. R. Ree, A family of simple groups associated with the simple Lie algebra G_2 , *Amer. J. Math.* 83, 1961, 432–462.
24. L. Ronyai, Zero divisors in quaternion algebras, *J. Algorithms* 9, 1988, 494–506.
25. R. Steinberg, Variations on a theme of Chevalley, *Pacific J. Math.* 9, 1959, 875–891.
26. M. Suzuki, On a class of double transitive groups I,II, *Ann. of Math.* 75, 1962, 105–145, 79, 1964, 514–589.

Coloring Hamming Graphs, Optimal Binary Codes, and the 0/1-Borsuk Problem in Low Dimensions

Günter M. Ziegler*

MA 7-1, Dept. Mathematics, TU Berlin, D-10623 Berlin, Germany

ziegler@math.tu-berlin.de,

<http://www.math.tu-berlin.de/~ziegler>

Abstract. The 0/1-Borsuk problem asks whether every subset of $\{0, 1\}^d$ can be partitioned into at most $d + 1$ sets of smaller diameter. This is known to be false in high dimensions (in particular for $d \geq 561$, due to Kahn & Kalai, Nilli, and Raigorodskii), and yields the known counterexamples to Borsuk's problem posed in 1933.

Here we ask whether there might be counterexamples in low dimension as well. We show that there is no counterexample to the 0/1-Borsuk conjecture in dimensions $d \leq 9$. (In contrast, the general Borsuk conjecture is open even for $d = 4$.)

Our study relates the 0/1-case of Borsuk's problem to the coloring problem for the Hamming graphs, to the geometry of a Hamming code, as well as to some upper bounds for the sizes of binary codes.

1 Introduction

The Borsuk conjecture is a puzzling problem: posed in 1933, in the famous paper by K. Borsuk [6] that contained the “Borsuk-Ulam theorem,” it has resisted all attempts of proof until in 1992 Kahn and Kalai [11] announced that the conjecture is false, due to counterexamples in dimensions 1325 and higher. After much subsequent work, we now know that the Borsuk conjecture is false in all dimensions $d \geq 560$, and true in dimensions $d \leq 3$ — which leaves a remarkable gap! How about dimension 4, say? This leads us to ask: “*Must the counterexamples be necessarily be so high-dimensional?*”

It turns out that while the proofs in dimensions $d \leq 3$ depend on intricate geometric arguments, all the counterexamples rely on purely combinatorial work on sets of 0/1-vectors plus some linear algebra techniques. Thus we ask: “*Must 0/1-counterexamples be necessarily be so high-dimensional?*” This question leads us to a lot of beautiful combinatorics, to graph coloring problems and optimal codes, and finally to a partial answer: Perhaps they don't have to be *that* high-dimensional, but at least there are no counterexamples in dimensions $d \leq 9$.

* Supported by a DFG Gerhard-Hess-Forschungsförderungspreis (Zi 475/2-3).

Acknowledgements. Please consider this paper a survey: There is very little new in it, it's just making connections between different facts, some of which are published, and some of which seem to be “well-known” (that is, well-known to those who well know them). I am reporting also about some contributions that I was shown by Lex Schrijver, Stefan Hougardy, Jon McCammond, and Noga Alon — thanks to them! I am also grateful to my ‘Diplomanden’ Jürgen Petersen [18] and Frank Schiller [22] for their contributions, and for many discussions.

2 The 0/1-Borsuk Problem

Borsuk asked:

Borsuk’s problem: Is it true that every bounded subset S of \mathbb{R}^d can be decomposed into $d + 1$ subsets,

$$S = S_1 \cup S_2 \cup \dots \cup S_{d+1},$$

all of which have smaller diameter than S ?

The number of $d+1$ subsets cannot be reduced: $d+1$ sets are needed, for example, if S is a regular simplex of dimension d (or just its vertex set), or a d -dimensional ball (or just its boundary). In both cases, however, a partition into $d + 1$ parts exists, and isn’t hard to find. (Only the part that a d -ball cannot be partitioned into fewer than $d + 1$ parts of smaller diameter is non-trivial; it is equivalent to the Borsuk-Ulam theorem, and was anticipated by Lusternik and Shnirel’man in 1930, three years before Borsuk’s paper.)

Borsuk’s conjecture was proved to be true for all sets S of dimension $d \leq 3$ (Perkal 1947; Eggleston 1955), and for smooth bodies S (Hadwiger 1946), but the general case remained an open problem for a long time. See Grünbaum [8] and Boltyanski, Martini & Soltan [5, §31] for surveys of Borsuk’s problem. On the other end, the constructions of Kahn & Kalai were simplified, extended and improved, so that with the efforts of Nilli [16], Raigorodskii [20,21] and Weißbach [24] we have counterexamples for all $d \geq 560$. (See [1] for a popular exposition.) Thus, with all the work and effort that was put into the problem, we know now that the answer is “yes” for $d \leq 3$ and “no” for $d \geq 560$.

Borsuk’s problem is hard enough for the special case where S is a finite set (equivalently, if one considers convex polytopes $\text{conv}(S)$, since the largest distance in a polytope always occurs between two vertices). It is an interesting question whether one can derive the general case from the polytope case.

An even more special case (but the one used to construct counterexamples!) is the one when S is a set of 0/1-vectors, that is, where $S \subseteq \{0, 1\}^d$ is a subset of the vertices of the regular 0/1-cube. In that special case, it is now known that Borsuk’s conjecture is false for $d \geq 561$, but true for $d \leq 9$. For the counterexamples in high dimensions, we refer to the sources quoted above; our aim in the following is to demonstrate the positive answer for the “0/1-Borsuk problem” for $d \leq 8$, and to explore some of the combinatorics, graph theory and coding theory connected with it.

0/1-Borsuk problem: For which d can every subset $S \subseteq \{0, 1\}^d$ be decomposed into $d + 1$ subsets,

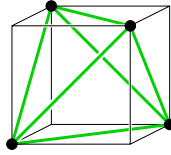
$$S = S_1 \cup S_2 \cup \dots \cup S_{d+1},$$

all of which have smaller diameter than S ?

It is not true that $d + 1$ subsets are even needed in each dimension: For example, in dimension 2 it is easy to check that 2 subsets will always do. However, the upper bound cannot be improved much.

Example 1. The subset $S := \{e_1, \dots, e_d\}$ is the vertex set of a regular $(d - 1)$ -dimensional simplex of edge length $\sqrt{2}$. This set can be decomposed into d subsets of smaller diameter, but not into fewer.

Example 2. A regular d -simplex with vertices in $\{0, 1\}^d$ exists if and only if there is a Hadamard matrix of order $d + 1$ (see [25]). For this $d + 1$ is necessarily equal to 1, to 2, or to a multiple of 4 (and it is conjectured that Hadamard matrices exist for all multiples of 4). Thus in dimensions d such that a Hadamard matrix of order $d + 1$ exists, we have an example of a subset S which needs $d + 1$ parts for its decomposition. Our figure displays a corresponding set for $d = 3$.



In the following, we'll treat the 0/1-Borsuk problem "case by case" in terms of two parameters; the first one is the dimension d , the second one is the integer k , with $1 \leq k \leq d$, such that \sqrt{k} is the diameter of the set $S \subseteq \{0, 1\}^d$ that we consider. Equivalently – and this will be useful in a coding theory context – the parameter just denotes the ℓ_1 -diameter, or Hamming diameter, of the set: for two 0/1-vectors \mathbf{x}, \mathbf{y} , the distance $\|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{k}$ is given by the number $k = \|\mathbf{x} - \mathbf{y}\|_1$ of coordinates in which \mathbf{x} and \mathbf{y} differ (by ± 1).

Thus, for every $d \geq 1$ and $1 \leq k \leq d$, we'll be studying the following problem:

Borsuk(d, k): Is it true that every subset $S \subseteq \{0, 1\}^d$ of diameter \sqrt{k} can be decomposed into $d + 1$ subsets,

$$S = S_1 \cup S_2 \cup \dots \cup S_{d+1},$$

all of which have smaller diameter than S ?

In this formulation, the 0/1-Borsuk problem is true in dimension d if and only if Borsuk(d, k) is true for all $k \in \{1, \dots, d\}$.

At this point, it is an instructive exercise to work out the 0/1-Borsuk problem for $d \leq 3$ – an exercise, however, that we leave to the interested reader.

3 Reformulation as a Coloring Problem

Assume that we are handed a particularly interesting set $S \subseteq \{0, 1\}^d$ of diameter \sqrt{k} , or just an example for which we are *told* that it is particularly interesting, and we are asked whether it satisfies Borsuk's conjecture, what should we do?

It seems that the problem is difficult, just because it is equivalent to a coloring problem, and coloring problems are difficult (in general), and we have not much indication that this one happens to be an easy special case.

Definition 1. We shall say that a subset $S \subseteq \{0, 1\}^d$ of (Euclidean) diameter \sqrt{k} has *Hamming diameter* k . It's *Borsuk graph* is the graph $B_k(S)$ with vertex set S , and with an edge between vertices $\mathbf{x}, \mathbf{y} \in S$ if the distance between \mathbf{x} and \mathbf{y} is the diameter of S (that is, if the Euclidean distance is \sqrt{k} , and the Hamming distance is k).

Now a partition of S into m subsets of smaller diameter is just the same thing as a partition of the vertex set of $B_k(S)$ into m stable subsets, that is, a coloring the graph $B_k(S)$ with m colors. This leads to the following "coloring version" of the 0/1-Borsuk problem. (This reduction to a coloring problem can be done the same way for the case of a general set $S \subseteq \mathbb{R}^d$, but it fails in the infinite case.)

Borsuk(d, k): Is it true that for every subset $S \subseteq \{0, 1\}^d$ of Hamming diameter k , the corresponding Borsuk graph has chromatic number

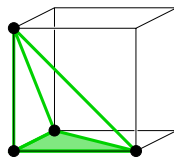
$$\chi(B_k(S)) \leq d + 1?$$

In this formulation, let's discuss (and get rid of) a number of simple cases:

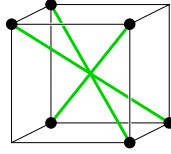
$k = 1$: If $k = 1$, then $|S| = 2$, $B_1(S) \cong K_2$, $\chi(B_1(S)) = 2 \leq d + 1$. No problem.

$k = 2$: 0/1-configurations of Hamming diameter 2 are easily classified, modulo 0/1-equivalence (cf. [25]), where we consider two sets as equivalent if we can transform one into the other using just symmetries of the 0/1-cube, that is, by permutation and complementation of coordinates.

The result is that there is one special case: the 3-dimensional regular tetrahedron discussed and depicted above, whose Borsuk graph is a K_4 , and thus the chromatic number is $\chi(B_2(S)) = \chi(K_4) = 4 \leq d + 1$. Any configuration that is not 0/1-equivalent to this one is 0/1-equivalent to a subset to the configuration $S = \{0, \mathbf{e}_1, \dots, \mathbf{e}_d\}$, whose Borsuk graph is $K_d + K_1$, a clique plus an isolated vertex; we get for this case that $\chi(B_2(S)) \leq d$. Thus Borsuk(d, k) holds for $k \leq 2$.



k is odd: In this case $B_k(S)$ is bipartite; we get a legal coloring by collecting the 0/1-vectors with odd coordinate sums, and the vertices with even coordinate sums, into two distinct classes. Thus here we have $\chi(B_k(S)) = 2$. Our figure shows such a case, where the points in S are the vertices of a 0/1-octahedron.



Thus we have dealt with the cases $k \leq 3$; the case $k = 4$ is already harder, see Section 9.2. Our next goal will be the situation where k is rather large (compared with d); it turns out that the bounds we get there are more generally valid for the “Hamming graphs,” which contain the Borsuk graphs as subgraphs.

4 Coloring Hamming Graphs

There are lots of different Borsuk graphs for any given d and k : but they are all subgraphs of the following “Hamming graphs.”

Definition 2. The *Hamming graph* $H_{d,k}$ has vertex set $\{0,1\}^d$, and two of its vertices are connected by an edge if and only if they have Hamming distance exactly k (that is, Euclidean distance \sqrt{k} .)

(We could have also used the notation $B_k(\{0,1\}^d)$ for $H_{d,k}$, but our convention is that the subscript k in $B_k(S)$ should denote the diameter of S .)

Lemma 1. *For every 0/1-set $S \subseteq \{0,1\}^d$ of diameter k we have*

$$\chi(B_k(S)) \leq \chi(H_{d,k}).$$

Proof. The Borsuk graph $B_k(S)$ is an induced subgraph of $H_{d,k}$. □

In particular, this means that we have proved $\text{Borsuk}(d, k)$ if we find out that $\chi(H_{d,k}) \leq d+1$. This holds for some parameters, while it is drastically false for others, as we shall see – it fails even for some d with $k = 2$, where we have already established that $\text{Borsuk}(d, k)$ is correct.

The following is a look at the Hamming graphs (and their chromatic numbers) for special parameters and examples.

k is odd: In this case $H_{d,k}$ is bipartite, and thus $\chi(H_{d,k}) = 2$, with the same argument as used above for the Borsuk graphs.

k is even: In this case a vertex with odd coordinate sum, and a vertex with even coordinate sum, cannot be connected by an edge. Thus the Hamming graph $H_{d,k}$ is disconnected for even k . The two components induced on the

even vertices and on the odd vertices are isomorphic, and can be identified with the graph

$$H_{d-1,\{k,k-1\}} = H_{d-1,k} \cup H_{d-1,k-1},$$

two of whose vertices are connected by an edge if their Hamming distance is either k or $k-1$. In particular, for $k=2$ the graph $H_{d-1,\{k,k-1\}}$ has an edge between any two distinct vertices of distance *at most* 2, and may be denoted by $H_{d-1,\leq 2}$.

k is large: If vertices $\mathbf{x}, \mathbf{y} \in \{0,1\}^d$ have distance k , then this means that they differ in k coordinates, and thus the first $d-k+1$ coordinates of \mathbf{x} and of \mathbf{y} cannot be all the same. This implies that

$$\mathbf{x} \longmapsto (x_1, \dots, x_{d-k+1}) \in \{0,1\}^{d-k+1}$$

is a legal coloring of $H_{d,k}$. The existence of this coloring implies that

$$\chi(H_{d,k}) \leq 2^{d-k+1}.$$

This bound is meaningful if $d-k$ is small; it is sharp if $d-k$ is very small. (It corresponds to the “Singleton bound” in coding theory.)

$k=d$: As a special case for the Singleton Bound, for $d=k$ we see that $H_{d,d}$ is a matching (of chromatic number 2).

$k=d-1$: The Singleton bound implies that the chromatic number of $H_{d,d-1}$ is at most 4. On the other hand, it is easy to see that for even $d-1$ the graph $H_{d,d-1}$ is not bipartite. Furthermore, a result of Payan [17] states that a “cubelike graph” such as $H_{d,d-1}$ cannot have chromatic number 3, so we find that $\chi(H_{d,d-1}) = 4$ for odd d , and $\chi(H_{d,d-1}) = 2$ for even d .

5 Some Coding Theory Bounds Are Used

The Hamming graphs $H_{d,2}$ have been studied extensively, also since they are unions of two components that are isomorphic to $H_{d-1,\leq 2}$. See for example [12] and [7], and especially the discussion in [10, Sect. 9.7] (and the references quoted there).

Let’s discuss upper and lower bounds for the chromatic numbers of these graphs independently.

Lemma 2. (Linial, Meshulam & Tarsi [12]) *For all $d \geq 1$,*

$$\chi(H_{d,2}) \leq 2^{\lceil \log_2(d) \rceil},$$

where the upper bound can be read as “ d rounded up to the next power of 2.”

Proof. Let $d \leq 2^m$, then an explicit 2^m -coloring can be given as follows. For each coordinate i ($1 \leq i \leq d$), let $b(i-1) \in \{0,1\}^m$ be the 0/1-vector of length m

obtained from the binary expansion of $i - 1$, adding leading zeroes as necessary. Then we color the vertices of $H_{d,2}$ by

$$c : \{0, 1\}^d \longrightarrow \{0, 1\}^m, \\ \mathbf{x} \longmapsto \sum_{i : x_i = 1} b(i - 1),$$

where the sum is taken “component-wise,” modulo 2. Thus if two vectors $\mathbf{x}, \mathbf{y} \in \{0, 1\}^d$ differ in exactly two coordinates i, j , then their colors $c(\mathbf{x})$ and $c(\mathbf{y})$ will differ exactly by $b(i - 1) + b(j - 1)$ (modulo 2), which is not zero since $i \neq j$. \square

According to MacWilliams & Sloane [14, Part II, p. 523], “Probably the most basic problem in coding theory is to find the largest code of a given length and minimum distance,” that is, the evaluation and estimation of the quantities given by the following definition.

Definition 3. For $d \geq 1$ and $1 \leq s \leq d$, $A(d, s)$ denotes the maximum number of codewords in any binary code of length d and minimum distance s between the codewords.

That is, $A(d, s)$ is the largest size of a subset $C \subseteq \{0, 1\}^d$ such that any two elements of C have Hamming distance at least s .

We refer to MacWilliams & Sloane [14, Chap. 17] for non-trivialities about these quantities, and their relevance. An excellent source for the “linear programming bounds” that are used to get non-trivial upper bounds (such as the ones used below) is [3]. As an example, we trivially have $A(d, 1) = 2^d$, and $A(d, d) = 2$.

Lemma 3. For all $t \geq 1$ and $d \geq 2t$, $A(d, 2t) = A(d - 1, 2t - 1)$.

Proof. Indeed, if we take any $(d, 2t)$ -code, then the operation of “deleting the last coordinate” yields a code of the same size (by $t \geq 1$) and minimum distance decreased by at most 1, and “adding a parity check-bit as a last coordinate” will take us from a $(d - 1, 2t - 1)$ -code to a $(d, 2t - 1)$ -code of the same size in which furthermore all code words and hence all distances are even, and which is thus a $(d, 2t)$ -code. \square

In particular, $A(d, 4) = A(d - 1, 3)$ holds for all $d \geq 4$. Now $A(d - 1, 3)$ is “by definition” also the largest size of an independent (“stable”) set in the graph $H_{d-1, \leq 2}$. Together with the fact that $H_{d,2}$ consists of two components that are isomorphic to $H_{d-1, \leq 2}$, this yields that the largest size of an independent set in $H_{d,2}$ is exactly $2A(d, 4)$.

At this point, we quote a result by Best & Brouwer [4] [3, p. 129] about shortened Hamming codes, which implies that

$$A(2^m - t, 4) = 2^{2^m - t - m - 1} \quad \text{for } 0 \leq t \leq 3.$$

This result, translated back to graph theory, says that the independent sets in the Hamming graphs $H_{d,2} = A(d, 4)$ “aren’t that large.” Thus it provides a

lower bound on the chromatic numbers via the inequality

$$\chi(G) \geq \frac{|V|}{\alpha(G)}$$

(where $\alpha(G)$ denotes the size of a largest independent set of vertices in G), which is valid for every finite graph G . Applied to $G = H_{d,2}$ for $d = 2^m - t$, this yields

$$\chi(H_{2^m-t,2}) \geq \frac{2^{2^m-t}}{2^{2^m-t-m-1}} = 2^m.$$

Thus we get the following result, which says that the upper bound of Lemma 2 is sharp for *some* values of d .

Proposition 1. (Linial, Meshulam & Tarsi [12]) *For all $d \geq 1$,*

$$\chi(H_{d,2}) \leq 2^{\lceil \log_2(d) \rceil},$$

with equality if d is of the form $d = 2^m$, $2^m - 1$, $2^m - 2$, or $2^m - 3$.

In particular, $\chi(H_{d,2}) = 2^{\lceil \log_2(d) \rceil}$ holds for all $d \leq 8$, and again for $13 \leq d \leq 16$. Of course this raises questions for the other values, in particular for $d = 9$. Let's increase the suspense a bit and postpone this question to Section 9.1.

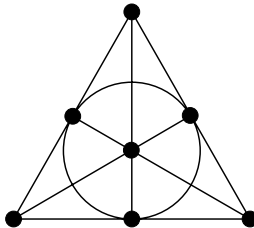
6 A Hamming Code Is Used

Payan [17] has proved that $\chi(H_{6,4}) \leq 7$ by exhibiting an explicit coloring. His claim that “it is not very difficult to prove” that the chromatic number is indeed 7 can be confirmed by computer [22]. Thus Payan disproved an earlier conjecture that “cube-like” graphs must necessarily have a chromatic number that is a power of 2.

In the following we will discuss that/why the chromatic numbers of both $H_{7,4}$ and $H_{8,4}$ are 8 — and we want to do that “by hand” (rather than leave it to a computer) since the geometry of the argument is so nice (due to Lex Schrijver [23]).

Lemma 4. $\chi(H_{7,4}) = \chi(H_{8,4}) = 8$.

Proof. We start with the binary Hamming code $H(3) \subseteq \{0,1\}^7$, which can be described as follows: Number the points of a Fano plane



by $1, 2, \dots, 7$, and take as the code words of your code the zero vector (corresponding to the empty subset), the all-ones vector (corresponding to the whole plane), the seven 0/1-vectors of weight 3 that correspond to the lines of the Fano plane, and the seven 0/1-vectors of weight 4 that correspond to their complements. Altogether this yields the 16 codewords of the Hamming code $H(3)$, about which the following facts are well-known and easily verified:

- $H(3)$ is a linear code (sums of codewords modulo 2 are codewords),
- it is a perfect code of minimum distance 3 (that is, every 0/1-vector of length 7 is either a code word, or it has distance 1 from a codeword),
- the complements of codewords are codewords as well.

Now the eight even code words, of weights 0 and 4 (corresponding to the empty set and to the complements of the lines in the Fano plane), all have distance 4 from each other, and thus we have found an 8-clique in $H_{7,4}$. At the same time, we can describe an 8-coloring of $H_{7,4}$ associated with this clique: for this we take eight colors for the eight code words in the clique, and give the same color to all the 0/1 vectors that have distance at most 1 either from the code word or from its complement. In other words, all vectors of distances 0, 1, 6 or 7 from an even code word get the same color, which yields a perfect 8-coloring.

To treat $H_{8,4}$, we use the extended Hamming code $\tilde{H}(3) \subseteq \{0, 1\}^8$, which is obtained by extending the code words of $H(3)$ by parity check bits. The resulting code is a linear code consisting of 16 code words, of minimum distance 4: indeed, all words other than the zero word and the all-ones word have weight 4. (Geometrically, this code corresponds to a remarkable 8-dimensional regular cross polytope of edge length $\sqrt{4} = 2$, whose vertices form a subset of the 0/1-cube.) Now the clique in $H_{7,4}$ consisted of the even code words, so it also determines an 8-clique in $H_{8,4}$. But it also yields an 8-coloring: for this we give the same color to all the 0/1-vectors that extend a vector of distance 0, 1, 6 or 7 from an even code word in $H(3)$. It is easily checked that no two vectors of Hamming distance 4 are assigned the same color by this rule. \square

7 Coloring Hamming Graphs, III

Let's collect the information that we have for the chromatic numbers of $H_{d,k}$ for $d \leq 9$ in a table:

$d \setminus k$	1	2	3	4	5	6	7	8	9
1	2								
2	2	2							
3	2	4	2						
4	2	4	2	2					
5	2	8	2	4	2				
6	2	8	2	7	2	2			
7	2	8	2	8	2	4	2		
8	2	8	2	8	2	≤ 8	2	2	
9	2	≤ 16	2		2	≤ 16	2	4	2

8 The 0/1-Borsuk Problem in Low Dimensions

The following table combines the upper bounds achieved in Section 3 with our knowledge about the chromatic numbers of Hamming graphs, which we have just summarized. The entry for d and k gives the best upper bound available for the maximal chromatic number of a Borsuk graph $B_k(S)$ for a subset $S \subseteq \{0, 1\}^d$ of Hamming diameter k :

$d \setminus k$	1	2	3	4	5	6	7	8	9
1	2								
2	2	2							
3	2	4	2						
4	2	4	2	2					
5	2	5	2	4	2				
6	2	6	2	7	2	2			
7	2	7	2	8	2	4	2		
8	2	8	2	8	2	8	2	2	
9	2	9	2		2		2	4	2

Theorem 1. *The 0/1-Borsuk problem has a “yes” answer for all $d \leq 9$.*

Proof. For $d \leq 8$, this follows from the fact that for $d \leq 8$ all the entries in the above table are at most $d + 1$.

The case $d = 9$ was done by Petersen [18], and relies on explicit coloring schemes that are rather complicated and will thus not be described or verified here. \square

9 Coloring Hamming Graphs, III

9.1 The Hamming Graphs $H_{d,2}$ – Stefan Hougardy

We had postponed the (interesting) case $d \geq 9$ of the graphs $H_{d,2}$. In particular, for $d = 9$ we can use that $A(8, 3) = A(9, 4) = 20$ [3], which yields a lower bound of $\lceil \frac{2^9}{2 \cdot 20} \rceil = 13$ for the chromatic number of $H_{9,2}$.

On the other hand, several of the better coloring heuristics do find a 14-coloring of $H_{9,2}$. Apparently this was first done by Hougardy in 1991, who got it from an adaption of the Petford-Welsh [19] algorithm. This leaves us with a rather narrow gap

$$13 \leq \chi(H_{9,2}) \leq 14.$$

The question whether the chromatic number of $H_{9,2}$ is 13 or 14 is a combinatorial covering problem: We are trying to cover the $2^8 = 256$ even vertices in the 9-cube by even $(9, 4)$ -codes. Hougardy [9] has found that there are only two non-equivalent such codes of the maximal size 20, but many more of sizes 19 or 18.

Now if a covering by 13 codes exist, then it must use at least 9 codes of size 20. Do they fit together?

More generally, if we want to get beyond the basic $\frac{|V|}{\alpha(G)} \leq \chi(G)$ lower bound for the chromatic number, then we must find out more about the “geometry” of the independent sets: these might be large enough, but they might not fit together to give a coloring with few colors.

For the higher values of $d = 10, 11, 12$, the available data are

$$\begin{array}{ll} A(9, 4) = 20 & 13 \leq \chi(H_{9,2}) \leq 14 \\ A(10, 4) = 40 & 13 \leq \chi(H_{10,2}) \leq 16 \\ 72 \leq A(11, 4) \leq 76 & 14 \leq \chi(H_{11,2}) \leq 16 \\ 144 \leq A(12, 4) \leq 152 & 14 \leq \chi(H_{12,2}) \leq 16 \\ A(13, 4) = 256 & \chi(H_{13,2}) = 16 \end{array}$$

The upper bounds for $A(11, 4)$ and $A(12, 4)$ are due to Litsyn & Vardy [13]; it is conjectured, however, that the lower bounds are tight [3, p. 128], which in turn would give a lower bound of 15 for $\chi(H_{11,2})$ and $\chi(H_{12,2})$. But a gap remains in either case ...

9.2 A Lower Bound for Small Diameter – Jon McCammond

Petersen [18] has shown that $\text{Borsuk}(d, 4)$ has a “yes” answer for all large enough d . And in fact, McCammond [15] has proved that for every fixed k the answer to $\text{Borsuk}(d, k)$ is positive when d is large enough (with respect to k).

Even stronger, it follows from the arguments and bounds obtained in [15] that $\text{Borsuk}(d, k)$ is true whenever $k \leq c\sqrt{\log d}$, for some constant $c > 0$. Thus, for counterexamples k can not be too small compared to d .

9.3 An Upper Bound for Large Diameter – Noga Alon

One can also show that for counterexamples to $\text{Borsuk}(d, k)$ the difference $d - k$ can not be too small when compared to d . For this, the “Singleton bound” that we had used in the case of very small $d - k$ is far from optimal.

Proposition 2. (Alon [2])

If $l \leq d$ is such that a Hadamard matrix $H_\ell \in \{1, -1\}^\ell$ exists, then

$$\chi(H_{d, > d - \sqrt{\ell}}) \leq 2\ell.$$

Proof. For simplicity, we describe this proof for vertex coordinates in $\{1, -1\}$. Let $\mathbf{v}_1, \dots, \mathbf{v}_\ell \in \{1, -1\}^\ell$ be the columns of H_ℓ , which form an orthogonal basis by definition. Thus the points $\{\pm \mathbf{v}_1, \dots, \pm \mathbf{v}_\ell\} \subseteq \{1, -1\}^\ell$ form the vertices of a regular cross polytope of edge length $\sqrt{2}\sqrt{\ell}$.

The points in $\{1, -1\}^\ell$ are now 2ℓ -colored according to the closest point from the set $\{\pm \mathbf{v}_1, \dots, \pm \mathbf{v}_\ell\}$. (In case of draws, decide arbitrarily.)

If an arbitrary vector $\mathbf{x} \in \{1, -1\}^\ell$ is expanded as $\mathbf{x} = \sum_{i=1}^\ell \lambda_i \mathbf{v}_i$ in terms of the orthogonal basis $\mathbf{v}_1, \dots, \mathbf{v}_\ell$, then one of the coefficients $\lambda_i = \frac{\langle \mathbf{x}, \mathbf{v}_i \rangle}{\langle \mathbf{v}_i, \mathbf{v}_i \rangle}$ has absolute value $|\lambda_i| \geq \frac{1}{\sqrt{\ell}}$, that is, \mathbf{x} has at least $\frac{1}{2}(\ell + \sqrt{\ell})$ components in common with the corresponding vector $\pm \mathbf{v}_i$. Thus two vectors in $\{1, -1\}^\ell$ that get the same color have at least $\sqrt{\ell}$ components in common.

From this we derive that if two vectors $\hat{\mathbf{x}}, \hat{\mathbf{y}} \in \{1, -1\}^d$ get the same color (according to their first ℓ coordinates), then they agree in at least $\sqrt{\ell}$ coordinates. \square

The point set used in this proof corresponds to a well-known structure in the $0/1$ -cube. Namely, the corresponding set

$$\mathcal{C}_\ell := \{\tfrac{1}{2}(\mathbf{1} \pm \mathbf{v}_1), \dots, \tfrac{1}{2}(\mathbf{1} \pm \mathbf{v}_\ell)\} \subseteq \{0, 1\}^\ell$$

is a binary code of length ℓ , minimum distance $\frac{1}{2}\ell$, consisting of 2ℓ code words, which form the vertices of a regular cross polytope of edge length $\sqrt{\frac{1}{2}\ell}$. It is known as the $(\ell, 2\ell, \frac{1}{2}\ell)$ -Hadamard code [14, Part I, p. 49].

In the special case $\ell = 8$, this Hadamard code is equivalent to the extended Hamming code $\tilde{H}(3)$ that we had used for a different purpose in Section 6. Thus we could have equivalently started the construction in Section 6 with the 8×8 Hadamard matrix instead of the Fano plane.

Together with the known existence results for Hadamard matrices (they are conjectured to exist for all $\ell = 4k$, and known to exist whenever $4k - 1$ is a prime power), Proposition 2 shows that Borsuk(d, k) is true whenever $d - k \leq c' \sqrt{d}$ for a constant $c' > 0$, which for large d can be taken to be arbitrarily close to $\frac{1}{2}\sqrt{2}$.

The counterexamples to Borsuk's conjecture due to Kahn & Kalai and the variations of Nilli, Raigorodski and Weißbach all have $k \sim \frac{d}{2}$.

References

1. M. Aigner & G. M. Ziegler: Proofs from THE BOOK, Springer-Verlag, Heidelberg Berlin 1998.
2. N. Alon: Personal communication, Oberwolfach 1999.
3. M. R. Best: Optimal codes, in: "Packing and Covering in Combinatorics" (A. Schrijver, ed.), *Mathematical Centre Tracts* **106** (1979), 119-140.
4. M. R. Best & A. E. Brouwer: The triply shortened binary Hamming code is optimal, *Discrete Math.* **17** (1977), 235-245.
5. V. Boltyanski, H. Martini & P. S. Soltan: Excursions into Combinatorial Geometry, Universitext, Springer-Verlag, Berlin Heidelberg 1997.
6. K. Borsuk: Drei Sätze über die n -dimensionale euklidische Sphäre, *Fundamenta Math.* **20** (1933), 177-190.
7. T. Dvořák, I. Havel, J.-M. Laborde & P. Liebl: Generalized hypercubes and graph embedding with dilation, *Rostocker Math. Kolloq.* **39** (1990), 13-20.
8. B. Grünbaum: Borsuk's problem and related questions, in: "Convexity" (V. Klee, ed.), Proc. Symposia in Pure Mathematics, Vol. VII, Amer. Math. Soc., Providence RI 1963, pp. 271-284.

9. S. Hougardy: personal and email communications, 1999/2000.
10. T. R. Jensen & B. Toft: Graph Coloring Problems, Wiley, New York 1994.
11. J. Kahn & G. Kalai: A counterexample to Borsuk's conjecture, *Bulletin Amer. Math. Soc.* **29** (1993), 60-62.
12. N. Linial, R. Meshulam & M. Tarsi: Matroidal bijections between graphs, *J. Combinatorial Theory, Ser. B* **45** (1988), 31-44.
13. Y. Litsyn & A. Vardy: The uniqueness of the Best code, *IEEE Transactions Information Theory* **40** (1994), 1693-1698.
14. F. J. MacWilliams & N. J. A. Sloane: The Theory of Error-Correcting Codes, Parts I and II, North Holland, Amsterdam 1977.
15. J. P. McCammond & G. M. Ziegler: The 0/1-Borsuk conjecture is generically true for each fixed diameter, Preprint 2000, in preparation.
16. A. Nilli: On Borsuk's problem, in: "Jerusalem Combinatorics '93" (H. Barcelo and G. Kalai, eds.), *Contemporary Mathematics* **178**, Amer. Math. Soc. 1994, 209-210.
17. C. Payan: On the chromatic number of cube-like graphs, *Discrete Math.* **103** (1992), 271-277.
18. J. Petersen: Färbung von Borsuk-Graphen in niedriger Dimension, Diplomarbeit, TU Berlin 1998, 39 pages.
19. A. D. Petford & D. J. A. Welsh: A randomised 3-colouring algorithm, *Discrete Math.* **74** (1989), 253-261.
20. A. M. Raigorodskii: On the dimension in Borsuk's problem, *Russian Math. Surveys* (6) **52** (1997), 1324-1325.
21. A. M. Raigorodskii: On a bound in Borsuk's problem, *Russian Math. Surveys* (2) **54**, 453-454.
22. F. Schiller: Zur Berechnung und Abschätzung von Färbungszahlen und der ϑ -Funktion von Graphen, Diplomarbeit, TU Berlin 1999, 50 pages.
23. A. Schrijver: personal communication, Barcelona 1997.
24. B. Weissbach: Sets with large Borsuk number, Preprint 1999; *Beiträge Geometrie Algebra*, to appear.
25. G. M. Ziegler: Lectures on 0/1-polytopes, in: "Polytopes — Combinatorics and Computation" (G. Kalai & G.M. Ziegler, eds.), *DMV-Seminar* **29**, Birkhäuser-Verlag Basel 2000, 1-44, in print.